VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2018 Daniel Chýlek

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Kompresní metoda Brotli
Brotli Compression Algorithm

2018 Daniel Chýlek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Daniel Chýlek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Kompresní metoda Brotli
Brotli Compression Algorithm

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je nastudovat a popsat kompresní metodu Brotli, a implementovat základní části algoritmu a provést experimentální ověření jejich funkčnosti.

Práce bude obsahovat:
1. Seznámení s problematikou komprese a metodou Brotli.
2. Podrobný popis jednotlivých částí algoritmu.
3. Implementaci jednotlivých části algoritmu.
4. Experimentální ověření funkčnosti a testování kompresní metody v porovnání s ostatními metodami.
5. Závěr - zhodnocení výsledků.

Seznam doporučené odborné literatury:

[1] RFC 7932, J. Alakuijala, Z. Szabadka, https://www.ietf.org/rfc/rfc7932.txt, 2016
[2] RFC 1951, P. Deutsch, https://www.ietf.org/rfc/rfc1951.txt, 1996.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Jan Platoš, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ U. Fakulta elektrotechniky a informatiky OSTRAVA

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

30.4.2018 .....
Datum odevzdání

..... *Podpis*

Chtěl bych poděkovat vedoucímu práce doc. Ing. Janu Platošovi, Ph.D. za odborné vedení a rady při zpracování této práce.

Abstrakt

Bakalářská práce se zabývá bezztrátovou kompresní metodou Brotli, která se specializuje na kompresi textových webových zdrojů a webových fontů. V teoretické části práce jsou vysvětleny principy na kterých je tato metoda stavěna a popsán algoritmus dekomprese, praktická část práce pak zahrnuje implementaci tohoto algoritmu v jazyce C#, srovnání Brotli s dalšími příbuznými kompresními metodami nad vlastním výběrem testovacích dat, a analýzu jednotlivých kompresních úrovní a jakým způsobem využívají jednotlivé součásti Brotli.

Klíčová slova

bezztrátová komprese, Brotli, Lempel-Ziv 77, Huffmanův kód

Abstract

This bachelor's thesis focuses on the Brotli compression algorithm, which is designed to compress web resources in plaintext form and web fonts. The theoretical part explains concepts and principles upon which Brotli is built, with a detailed description of the decompression algorithm. The practical part then comprises an implementation of Brotli decompression in the C# language, a comparison to related compression methods using a custom selection of test data, and an analysis of Brotli's compression levels and how they utilize the individual features of Brotli.

Key words

lossless compression, Brotli, Lempel-Ziv 77, Huffman coding

Obsah

Seznam použitých symbolů a zkratek

LZ77	*Lempel-Ziv 77*, bezztrátový kompresní algoritmus který používá techniky ze kterých vychází další algoritmy, včetně Brotli
LSB	*Least Significant Bit*, nejméně významný bit čísla v binární soustavě
MSB	*Most Significant Bit*, nejvýznamnější bit čísla v binární soustavě
RTF	*Rich Text Format*, známý formát dokumentů podporující jednoduché formátování
PDF	*Portable Document Format*, také známý formát dokumentů založen na PostScriptu
HTML	*Hypertext Markup Language*, značkovací jazyk pro tvorbu webových stránek
CSS	*Cascading Style Sheets*, jazyk pro popis stylu zobrazení prvků webových stránek
JS	*JavaScript*, skriptovací jazyk běžící na straně prohlížeče webových stránek
PNG	*Portable Network Graphics*, obrázkový formát využívající bezztrátovou kompresi algoritmem DEFLATE
WOFF	*Web Open Font Format*, formát pro fonty který od verze 2.0 využívá kompresi Brotli
UTF-8	*Unicode Transformation Format*, způsob kódování textu který ve variantě UTF-8 kóduje znaky do 1-4 bajtů; jde o nejpoužívanější způsob kódování webových souborů

KiB	kibibyte (1024^1 bajtů)
MiB	mebibyte (1024^2 bajtů)
Mbit	megabit ($1000^2 \div 8$ bajtů)

$[a, b]$	uzavřený interval, pokud není řečeno jinak pak a i b jsou celá čísla
$(a, b, c, ...)$	pole prvků (lineární kolekce pevné délky)

Seznam tabulek

Seznam ilustrací

1. Úvod

Komprese dat je proces kterým se snižuje jejich objem, což šetří místo na úložném zařízení jako je například pevný disk nebo CD, a umožňuje rychlejší přenos dat po síti.

Rozlišujeme mezi bezztrátovou kompresí umožňující přesnou rekonstrukci originálních dat, vhodnou například pro text, obrázky, nebo archivy s více soubory, a ztrátovou kompresí při které jsou některé informace ztraceny nebo zkresleny, zpravidla tak aby byl rozdíl co nejméně patrný, což je vhodné například pro fotografie, video, a audio.

Komprese je nepostradatelnou součástí dnešních technologií, také z důvodu rychlého rozšiřování internetu do celého světa – nejen by bez komprese nemohly některé internetové služby existovat, například streaming videí milionům lidí denně, ale také výrazně snižuje množství dat které je potřeba přenést při načítání webových stránek, a další internetové komunikaci kterou využívá neustále se zvyšující počet zařízení.

Tato práce se konkrétně zaměřuje na Brotli, což je jednou z novějších metod pro bezztrátovou kompresi a dekompresi primárně textových dat, se specifickým zaměřením na webové technologie. Cílem práce je popis formátu komprimovaného datového proudu, popis a implementace algoritmu dekomprese v jazyce C#, srovnání Brotli s dalšími algoritmy které mají podobné zaměření, a analýza využití a efektivity jednotlivých součástí Brotli.

V první kapitole *Brotli a jeho součásti* budou popsány důležité principy na kterých je Brotli stavěn, a které jsou potřebné pro jeho pochopení jako celku.

Následující kapitola *Specifikace formátu Brotli a algoritmus dekomprese* popíše algoritmický postup dekódování jednotlivých součástí Brotli, a spojí je dohromady v konzolové aplikaci která bude umět jakýkoliv datový proud komprimovaný metodou Brotli dekomprimovat.

Nakonec v kapitole *Testování Brotli a dalších algoritmů* bude s pomocí knihoven třetích stran do konzolové aplikace přidána podpora komprese a dekomprese několika dalšími metodami pro bezztrátovou kompresi textu, a tyto metody budou porovnány s metodou Brotli v úspoře místa a rychlosti komprese/dekomprese. Také budou analyzovány jednotlivé součásti Brotli, a jak jsou ovlivněny různými úrovněmi komprese a typy komprimovaných souborů.

2. Brotli a jeho součásti

Dokument *RFC 7932*[1] popisuje jednotlivé součásti formátu, jejich zápis v bitovém proudu s postupem dekódování, a pseudokód algoritmu pro dekompresi.

Brotli je kombinací bitově i znakově orientovaného formátu, kde znak má velikost 8 bitů. Zdrojová data (soubor, odpověď ze serveru) většinou čteme po jednotlivých bitech nebo sekvencích bitů které mohou být menší než jeden znak, nastávají ovšem případy kdy je potřeba přečíst sekvencí znaků.

Znaky vždy začínají na pozici která je násobkem 8, proto když přecházíme ze čtení po bitech na čtení po znacích, musíme nejdříve aktuální pozici v proudu bitů zarovnat na násobek osmi – pokud jsme například dosud přečetli 20 bitů, tak přeskočíme 4 bity abychom se dostali na pozici 24.

Každý datový proud komprimovaný metodou Brotli začíná hlavičkou obsahující parametry platící pro celý proud, a pokračuje jedním nebo více meta-bloky. Každý meta-blok se dále skládá z hlavičky a dat, kde hlavička obsahuje informace a struktury potřebné pro dekompresi dat daného meta-bloku.

V následujících podkapitolách si vysvětlíme důležité součásti a koncepty ze kterých Brotli vychází, a které jsou nutné pro pochopení formátu jako celku.

2.1. Posuvné okénko

Brotli, stejně tak jako řada dalších kompresních algoritmů založených na LZ77, používá posuvné okénko. Princip posuvného okénka spočívá v tom, že během dekomprese je v paměti uchován konstantní počet nejnověji dekomprimovaných bajtů, na něž se lze odkazovat tzv. zpětnými referencemi. „LZ77 těží z faktu, že slova a fráze se v textovém souboru často opakují. Opakovaný text lze zapsat jen jako odkaz na jeho dřívější výskyt, společně s počtem shodných znaků.“[2]

U optimalizovaného dekompresního algoritmu tak není využití paměti závislé na velikosti vstupního souboru, jelikož dekomprimované bajty mimo posuvné okénko mohou být zapsány do výstupního proudu (např. do souboru na disku), a z vyrovnávací paměti odstraněny.

Brotli podporuje posuvná okénka o velikostech přibližně 1 KiB až 16 MiB. Velikost okénka je dána v hlavičce s parametry platící pro celý datový proud.

2.2. Huffmanovo kódování

Huffmanovo kódování se používá pro bezztrátovou kompresi textu. Jednotlivé symboly dané abecedy jsou zakódovány jako sekvence bitů různých délek tak, aby nejčastěji používané symboly byly zakódovány nejkratšími sekvencemi bitů. Huffmanův kód je typ prefixového kódu, je tedy možné každý symbol jednoznačně dekódovat aniž by mezi nimi byly oddělovače.

---

1 ALAKUIJALA, Jyrki a Zoltan SZABADKA. RFC 7932. *IETF Tools* [online]. 2016 [cit. 2018-04-24]. Dostupné z: https://tools.ietf.org/html/rfc7932

2 CHOUDHARY, Suman M., Anjali S. PATEL a Sonal J. PARMAR. Study of LZ77 and LZ78 Data Compression Techniques. *International Journal of Engineering Science and Innovative Technology* [online]. 2015, **4**(3) [cit. 2018-04-24]. ISSN 2319-5967. Dostupné z: https://pdfs.semanticscholar.org/e087/085514d51ed377eec77b7a75c211d8739231.pdf

Pro dekódování proudu bitů je potřeba znát použitý Huffmanův kód. Ten si můžeme představit jako plný binární strom, tj. každý vrchol je buď listem nebo z něj vychází přesně dva potomci reprezentující 0 a 1. Pro dekódování začneme v kořeni a chodíme vždy do uzlu vlevo (0) nebo vpravo (1), dokud se nedostaneme do listu s dekódovaným symbolem. Symbol zapíšeme a vracíme se do kořene.

Brotli používá tzv. *kanonický Huffmanův kód*, který každému symbolu přiřadí pouze délku sekvence bitů která daný symbol reprezentuje[3], což umožňuje velmi kompaktní uložení struktury stromu. Při rekonstrukci postupujeme od nejkratších sekvencí, a začínáme vždy nulou. Pokud např. máme seznam délek (1, 2, 3, 3), umíme jednoznačně určit jejich sekvence bitů (0, 10, 110, 111), a pak k nim jen stačí přiřadit symboly.

2.3. Příkazy

Komprimovaná data jsou složena z tzv. *insert&copy* příkazů, které slouží k rekonstrukci původních dat. V každém příkazu se používají tři kategorie kódů:

- **Literály**
  - Každý literál je hodnota v intervalu [0, 255] a generuje jeden bajt dekomprimovaných dat
- **Insert&copy délky**
  - Jedná se o kód s hodnotou v intervalu [0, 703] ve které jsou zakódovány informace o páru délek *insert length* a *copy length*, a pravdivostní hodnota *distance code zero* která upravuje čtení vzdálenosti
  - Délka *insert length* udává počet literálů v tomto příkazu
  - Délka *copy length* udává buď počet znaků které jsou zkopírovány ze zpětné reference, nebo délku slova ve statickém slovníku; tato délka se zároveň používá pro kontextové modelování vzdáleností (viz níže)
- **Vzdálenosti**
  - Kód vzdálenosti je hodnota s rozmezím závislým na parametrech v hlavičce meta-bloku, a určuje buď konkrétní hodnotu vzdálenosti (v některých případech je potřeba přečíst další bity ze vstupního proudu), nebo odkaz na jednu ze 4 předchozích použitých hodnot vzdálenosti
  - Pokud je parametr *distance code zero* pravdivý, pak je čtení vzdálenosti přeskočeno a použije se předchozí použitá hodnota vzdálenosti
  - Hodnota vzdálenosti určuje buď začátek zpětné reference, anebo pokud je vzdálenost větší než velikost posuvného okénka nebo počet dosud dekomprimovaných bajtů, pak je v ní zakódována reference na slovo ve statickém slovníku

3 MCCLOSKEY, Robert. Canonical Huffman Coding. *CMPS 340* [online]. 2015 [cit. 2018-04-24]. Dostupné z: http://www.cs.uofs.edu/~mccloske/courses/cmps340/huff_canonical_dec2015.html

Každý příkaz *insert&copy* začíná přečtením insert&copy délek, následují literály, a nakonec vzdálenost. Pokud je tento příkaz poslední, může být ukončen okamžitě po zpracování literálů v případě, že jsou v tom momentu již všechna data meta-bloku dekomprimována (neboli pokud je velikost dosud přečtených dat rovna velikosti dat definované v hlavičce meta-bloku).

Hodnoty každé z těchto tří kategorií jsou zakódovány pomocí Huffmanových kódů. Zde ovšem nastává menší komplikace – kde například DEFLATE má jeden Huffmanův kód pro symboly a jeden pro vzdálenosti, Brotli umožňuje použití i stovek Huffmanových kódů v jednom souboru, čehož je dosaženo kombinací dvou spolu souvisejících principů:

- Rozdělení každé kategorie v rámci jednoho meta-bloku do 1 nebo více skupin (ve specifikaci formátu jsou skupiny označeny jako *block type*)
- Kontextové modelování pro literály a vzdálenosti

2.3.1. Rozdělení kategorií na skupiny

V rámci jednoho meta-bloku může být každá kategorie (literály, insert&copy délky, vzdálenosti) rozdělena na až 256 různých skupin. Brotli umožňuje přepínání aktuální skupiny před každým přečtením kódu jakékoliv z těchto kategorií. Skupiny se mohou libovolně opakovat, na každou skupinu se lze odkazovat indexem v intervalu [0, 255].

Počet skupin pro každou kategorii je definován v hlavičce meta-bloku. Pokud je počet skupin alespoň 2, následuje definice dvou Huffmanových kódů které jsou použity v každém *block-switch* příkazu dané kategorie – jeden z nich určí následující skupinu (tj. její index), a ten druhý určí počet zbývajících hodnot této skupiny (tento Huffmanův kód je okamžitě použit pro určení počtu zbývajících hodnot první skupiny).

Před přečtením kódu každé kategorie zkontrolujeme počet zbývajících hodnot aktuální skupiny. Pokud je tento počet 0, okamžitě je přečten příkaz *block-switch* (dva zmíněné Huffmanovy kódy) který změní aktuální skupinu. Po přečtení kódu je počet zbývajících hodnot snížen o 1.

Každá skupina obsahuje jeden nebo více Huffmanových kódů – literály mohou mít až 64 různých Huffmanových kódů, vzdálenosti mohou mít až 4, insert&copy délky pouze 1. Který Huffmanův kód ve skupině je před každým čtením kódu kategorie použit určuje kontextové modelování.

2.3.2. Kontextové modelování

Kontextové modelování v textové kompresi je princip, kdy způsob jakým je zakódován další znak závisí na jednom nebo více předchozích znacích. Pokud hovoříme o kontextovém modelování řádu N, pak je použito N předchozích znaků.

Brotli používá tuto definici kontextového modelování druhého řádu pro literály, zatímco pro vzdálenosti je použit upravený princip s kontextovým modelem závislým na hodnotě *copy length*.

Pro obě kategorie hlavička meta-bloku definuje *kontextovou mapu* právě tehdy, když má daná kategorie 2 nebo více Huffmanových kódů. Kontextová mapa je pole o velikosti (*počet skupin* × *počet kódů na skupinu*), a obsahuje indexy Huffmanových kódů pro danou kategorii. Počet kódů na

skupinu je vždy 64 pro literály a 4 pro vzdálenosti, proto nám pro nalezení indexu stačí jen aktuální skupina a posunutí (*offset*).

Literály

Každá skupina kategorie literálů se může odkazovat na až 64 různých Huffmanových kódů. Který z nich je použit závisí na kontextovém módu.

Kontextový mód je funkce, která jako vstup přijímá poslední dva dekomprimované bajty, a vrací hodnotu v intervalu [0, 63] která udává posunutí v kontextové mapě. Každá hlavička meta-bloku pro každou skupinu literálů definuje jeden z následujících kontextových módů:

- **LSB6**
  - Vrací hodnotu 6 nejméně významných bitů posledního dekomprimovaného bajtu
- **MSB6**
  - Vrací hodnotu 6 nejvíce významných bitů posledního dekomprimovaného bajtu
- **UTF8**
  - Použije hodnoty posledních dvou dekomprimovaných bajtů pro výpočet následujícího kontextu s pomocí vyhledávací tabulky, která je s definicí funkce uvedena v příloze
  - Mód je optimalizovaný pro kompresi textu
- **Signed**
  - Použije hodnoty posledních dvou dekomprimovaných bajtů pro výpočet následujícího kontextu s pomocí vyhledávací tabulky, která je s definicí funkce uvedena v příloze
  - Mód je optimalizovaný pro kompresi binárních dat

Vzdálenosti

Každá skupina kategorie vzdáleností se může odkazovat na až 4 různé Huffmanovy kódy. Princip kontextového modelování v případě vzdáleností je odlišný od literálů, protože pro výpočet následujícího kontextu namísto předchozí vzdálenosti používá délku *copy length* aktuálního příkazu:

Tabulka 1: Vztah hodnoty copy length s kontextovým modelem pro vzdálenosti

Hodnota *copy length*	2	3	4	5 a více
Posunutí v kontext. mapě	0	1	2	3

Tento model umožňuje použít jiné Huffmanovy kódy pro velmi krátké zpětné reference, a jiné pro dlouhé zpětné reference a slovníková slova. Je mnohem pravděpodobnější že najdeme stejnou kombinaci dvou bajtů blízko u sebe, zatímco identické kombinace např. 20 bajtů jsou často velmi vzdálené.

2.4. Statický slovník

Brotli využívá 120 KiB statický slovník, který „obsahuje 13 504 slov a slabik z angličtiny, španělštiny, čínštiny, hindštiny, ruštiny, arabštiny, a také některých strojově čitelných jazyků, především HTML a JavaScriptu.“[4]

Slova jsou seřazena do skupin podle jejich délky, která je v rozmezí [4, 24] bajtů. Jelikož známe počet slov v každé skupině, tak nám na referenci konkrétního slova stačí jen jeho délka a index ve skupině.

Každé slovo lze transformovat jednou ze 121 možných transformací. Každá transformace se skládá z prefixu který je přidán před slovo, sufixu který je přidán za slovo, a funkce která transformuje samotné slovo. Seznam všech transformací je uveden v příloze.

Pokud je při dekompresi dat přečtena zpětná reference která odkazuje mimo posuvné okénko (tj. vzdálenost je vyšší než velikost posuvného okénka nebo počet dosud dekomprimovaných bajtů), pak se jedná o referencí na slovo ve slovníku. Hodnota *copy length* určuje délku slova, a v hodnotě vzdálenosti je zakódován index slova a transformace.

2.4.1. Transformační funkce

Identity
Neprovede žádnou transformaci. Používá se pro transformace které pouze přidávají prefix a/nebo sufix, a samotné slovo nemění.

OmitFirst(N)
Vynechá prvních N znaků slova, kde N je v rozmezí [1, 9]. Jelikož všechna slova mají pevnou délku, tato funkce je použita pro extrahování jeho části.

OmitLast(N)
Vynechá posledních N znaků slova, kde N je v rozmezí [1, 9].

FermentFirst
Aplikuje funkci *Ferment*, popsanou níže, na první kódový bod UTF-8 slova.

FermentAll
Aplikuje funkci *Ferment* na každý kódový bod UTF-8 slova, se speciálním případem kde 4bajtové kódové body jsou považovány za 3bajtové kódové body.

2.4.2. Funkce Ferment

Brotli definuje funkci *Ferment* která identifikuje velikost kódového bodu UTF-8 na dané pozici ve slově, a transformuje jej. Připomeňme si, že velikosti kódových bodů se pohybují v rozmezí 1-4 bajtů. Obecně je cílem této funkce převod malých písmen na velká, ale neplatí to tak vždy.

---

4 ALAKUIJALA, Jyrki, Evgenii KIUCHNIKOV, Zoltan SZABADKA a Lode VANDEVENNE. Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms. *GitHub* [online]. Google, 2015 [cit. 2015-09-22]. Dostupné z: https://github.com/google/brotli/blob/master/docs/brotli-comparison-study-2015-09-22.pdf

Funkce provede bitovou operaci XOR (exkluzivní disjunkci nad všemi bity dvou čísel, respektive bajtů), jejíž parametry se liší v závislosti na velikosti kódového bodu.

1 bajt
Pokud znak reprezentuje malé písmeno anglické abecedy (a-z), provede nad hodnotou tohoto bajtu operace XOR 00100000, což převede malé písmeno a-z na velké písmeno A-Z.

2 bajty
Provede nad hodnotou 2. bajtu operaci XOR 00100000. K dvoubajtovým kódovým bodům UTF-8 se řadí několik skupin:

- Latin-1 Supplement (kontrolní kódy, interpunkce a symboly, některé znaky latinské abecedy s diakritikou)
- Mezinárodní fonetická abeceda
- Kombinující diakritické znaky
- Cyrilice
- Řecká, hebrejská, arabská, arménská, a další abecedy

Pokud se podíváme pouze na diakritiku Latin-1 Supplement, a některé znaky cyrilice a řecké abecedy, najdeme příklady kde tato operace převede malé písmeno na velké a naopak. Kompresor ovšem v tomto případě nijak negarantuje, že provedení operace nad znakem dává sémantický smysl.

Například slovo *Français* je správně převedeno na *FRANÇAIS*, ale slovo *Čeština* jehož diakritické znaky nejsou v Latin-1 Supplement je převedeno na *ČEŁTINA*.

3 bajty
Provede nad hodnotou 3. bajtu operaci XOR 00000101. Podle vývojářů byla tato operace vybrána experimentálně a nemá žádný sémantický smysl.[5]

4 bajty
Případ pro 4bajtové kódové body je zpracováván stejně jako případ pro 3bajtové kódové body. Transformační funkce *FermentAll* při procházení slova také postoupí o 3 bajty namísto 4.

I když se technicky jedná o chybu, ve skutečnosti tento případ téměř nikdy nenastane protože slovník neobsahuje žádné existující 4bajtové UTF-8 znaky. Slovník ovšem obsahuje několik slov s různými sekvencemi bajtů s vysokými hodnotami, u kterých tento případ může nastat.

5 Zdrojem informace je odpověď jednoho z vývojářů Brotli na můj e-mail s otázkami ohledně formátu.

3. Specifikace formátu Brotli a algoritmus dekomprese

3.1. Vlastní implementace

Dříve než začneme s implementací algoritmu dekomprese v jazyce C# a prostředí .NET Framework je potřeba rozhodnout se jakým způsobem budou části algoritmu rozděleny do tříd, a implementovat několik datových struktur.

V .NET Frameworku jsou pro reprezentaci a přenos binárních dat zpravidla použity proudy (*Stream*) které podporují operaci zápisu a/nebo čtení. Proudy které při zápisu nebo čtení data transformují, k čemuž řadíme například dekompresi, jsou založeny na vzoru *dekorátor* – existující rozhraní .NET Frameworku pro dekompresi přijímají komprimovaný proud, a podporují operaci čtení která v mezipaměti generuje dekomprimovaná data. Proudy jsou typicky navrženy tak, aby uměly operovat nad menšími kusy dat, což je vhodné např. při práci s velkými soubory na disku, ovšem ve vlastní aplikaci dekompresoru z důvodu jednoduchosti proud budeme a dekomprimovat všechna vstupní data najednou.

Bitový proud

Nejdůležitější datovou strukturou je bitový proud, který musí umět číst samostatné bity, skupiny bitů, a také zarovnání pozice na násobek 8 při přechodu na čtení po znacích a následné čtení celých bajtů. Pro reprezentaci dat bude použit lineární spojový seznam 64bitových celých čísel, který lze zkonstruovat z pole bajtů, a vlastní implementace iterátoru (*Enumerator*) která uchovává aktuální pozici, pohybuje se v seznamu 64bitových čísel, a provádí nad nimi operace pro extrakci bitů.

Dále bude použit tzv. *prostředník*, který zpřístupní jednodušší rozhraní pro práci s iterátorem bitů, včetně podpory čtení skupin bitů a jejich převod na celé číslo. Na něm budou závislé třídy, které se starají o čtení a dekódování jednotlivých součástí formátu Brotli.

Cyklická fronta

Jelikož .NET Framework neobsahuje existující implementaci cyklické fronty, bude použita vlastní implementace založena na poli s ukazatelem na nejnovější hodnotu, což v tomto případě stačí – někdy jsou cyklické fronty implementovány s jedním ukazatelem pro čtení a jedním ukazatelem pro zápis, zde to není potřeba.

Cyklická fronta podporuje operaci vložení nové hodnoty, a přečtení nejnovější hodnoty, druhé nejnovější hodnoty, atd. až do nejstarší hodnoty která bude příštím vložením přepsána.

Huffmanovy stromy

Huffmanův strom obsahuje dva typy vrcholů, které budou reprezentovány dvěma třídami – větev ze které vycházejí dvě cesty (jedna cesta pro 0, druhá cesta pro 1), a list ve kterém je uložen znak.

Obě třídy použijí společné rozhraní s operací přečtení znaku, která jako vstup přijímá iterátor bitů. Pokud je volaným objektem větev, přečte z iterátoru jeden bit který určí jeden ze dvou vrcholů větve, a na něm bude opět zavolána operace přečtení znaku s novým stavem iterátoru. Pokud je volaným objektem list, pouze vrátí uložený znak.

Pro ilustraci toho, jak Huffmanovo kódování funguje, aplikace umí vytvořit Huffmanův strom ze vstupního textu, zakódovat jej, a zobrazit informace o vygenerovaném stromu a zmenšení objemu dat. Způsob konstrukce Huffmanových stromů konkrétně v Brotli bude popsán v další kapitole.

Organizace projektu
Čtení a dekódování jednotlivých částí formátu Brotli budou zajišťovat především samostatné statické třídy, každá s veřejnou metodou která použije prostředníka pro čtení bitů, a vrátí příslušná data. Tato organizace přibližně kopíruje strukturu práce, kde každá podkapitola této části se vždy zaměřuje na přečtení a dekódování jedné konkrétní části formátu.

Pro reprezentaci meta-bloku bude použita třída, která bude inicializována všemi daty hlavičky, a bude umět přečíst datovou část meta-bloku. Každému meta-bloku bude také předán stavový objekt, sloužící k uchování sdílených informací, a také ke zprostředkování dat vygenerovaných předchozími meta-bloky jež jsou potřeba pro zpětné reference.

3.2. Obecné kódy

Brotli obsahuje dva obecné kódy které jsou použity na více místech najednou.

3.2.1. Kód s proměnnou délkou 1-11 bitů

Tento kód umožňuje přečíst hodnotu v intervalu [1, 256]. Pokud bychom použili kód s pevnou délkou, tak bychom potřebovali pro zápis každé možné hodnoty přesně 8 bitů, tento kód je ovšem optimalizovaný tak, aby nižší hodnoty potřebovaly mnohem méně bitů než vyšší hodnoty. Vycházíme z předpokladu, že nižší hodnoty se v místech kde je tento kód využit vyskytují mnohem častěji než vyšší hodnoty, a tudíž je v tomto případě kompaktnější než kód s pevnou délkou.

Postup přečtení kódu je následující:

1. Přečteme jeden bit, pokud je nulový pak je výsledná hodnota 1 a dále nepokračujeme
2. Přečteme tři bity, jejich hodnotu označíme *n*
3. Přečteme *n* následujících bitů, jejich hodnotu označíme *x*
4. Výsledná hodnota je $(2^n + x + 1)$

Pro referenci můžeme použít následující tabulku. Pro jednodušší pochopení jsou bity rozděleny do skupin, bity uvnitř každé skupiny čteme zprava doleva (tj. [1][011][100] je ve skutečnosti 1110001):

Tabulka 2: Mapování posloupností bitů na hodnoty kódu s proměnnou délkou 1-11 bitů

Posloupnost bitů	Rozmezí hodnot
[0]	1
[1][000]	2
[1][001][x]	[3, 4]
[1][010][xx]	[5, 8]
[1][011][xxx]	[9, 16]
[1][100][xxxx]	[17, 32]
[1][101][xxxxx]	[33, 64]
[1][110][xxxxxx]	[65, 128]
[1][111][xxxxxxx]	[129, 256]

3.2.2. Huffmanův kód

Pro přečtení seznamu délek a přiřazení symbolů rekonstruovanému Huffmanovu kódu Brotli používá dva vlastní způsoby: *jednoduchý* a *komplexní*. Nejdříve přečteme 2 bity a rozhodneme se podle jejich hodnoty:

- Hodnota 1 znamená použití jednoduchého kódu
- Hodnoty 0, 2, 3 znamenají použití komplexního kódu, s touto hodnotou jako parametrem

Jednoduchý kód

Jednoduchý kód je použit v případě, že je potřeba reprezentovat jen 1-4 symboly. Brotli 5 možných seznamů délek sekvencí bitů, které budou použity při konstrukci kanonického Huffmanova kódu:

- **1 symbol:** (0)
- **2 symboly:** (1, 1)
- **3 symboly:** (1, 2, 2)
- **4 symboly:** (2, 2, 2, 2) nebo (1, 2, 3, 3)

Počet symbolů je dán dalšími 2 bity, k hodnotě je pak přičtena 1 což nám dá hodnotu v rozmezí [1, 4]. Následují samotné symboly. Velikost každého symbolu lze odvodit z velikosti abecedy – pokud např. máme abecedu o 704 symbolech, pro reprezentování všech možných symbolů potřebujeme alespoň 10 bitů. Pokud tedy chceme přečíst 4 po sobě jdoucí symboly z této abecedy, přečteme ze vstupního proudu celkem (4 × 10) bitů. Formát dále vyžaduje, aby byly přečtené symboly před přiřazením délek seřazeny podle jejich pozice v abecedě.

Pokud je použit 1 symbol, tedy jeho sekvence bitů má délku 0, je zkonstruován Huffmanův strom obsahující pouze list, který při přečtení vrátí daný symbol a ze vstupního proudu nic nečte.

Pokud jsou použity 4 symboly, kde jsou dva možné seznamy délek, pak za symboly následuje 1 bit určující který seznam bude použit – hodnota 0 použije (2, 2, 2, 2), hodnota 1 použije (1, 2, 3, 3).

Komplexní kód

Komplexní kód rekonstruuje Huffmanův kód tak, že prochází všechny symboly abecedy, a každému přiřazuje délku bitové sekvence která tento symbol reprezentuje.

Při čtení komplexního kódu jsou použity speciální kódy s hodnotami v intervalu [0, 17]:

- 0 je nulová délka, symbol tedy není použit
- 1 až 15 reprezentuje délku bitové sekvence s touto hodnotou
- 16 je speciální kód který několikrát opakuje předchozí nenulovou délku, nebo délku 8 pokud je tento kód 16 první
    - Následují 2 bity které určí kolikrát předchozí délku opakovat (3-6 krát)
    - Lze jej řetězit – každý následující kód 16 v řetězu upravuje předchozí počet opakování *x*: x = (4 × (x - 2)) + 3 + (*hodnota 2 následujících bitů*)
- 17 je speciální kód který několikrát opakuje nulovou délku, neboli tyto symboly přeskočí
    - Následují 3 bity které určí kolikrát nulovou délku opakovat (3-10 krát)
    - Lze jej řetězit – každý následující kód 17 v řetězu upravuje předchozí počet opakování *x*: x = (8 × (x - 2)) + 3 + (*hodnota 3 následujících bitů*)

Tyto speciální kódy jsou samotny zakódovány ještě dalším Huffmanovým kódem, který je napevno určen ve specifikaci Brotli. Abeceda s těmito speciálními kódy je definovaná jako pole s hodnotami (1, 2, 3, 4, 0, 5, 17, 6, 16, 7, 8, 9, 10, 11, 12, 13, 14, 15). Tuto abecedu postupně procházíme, a každému speciálnímu kódu přiřazujeme délku jeho bitové sekvence. Délky jsou uloženy v pevně daném Huffmanovém kódu:

Tabulka 3: Mapování posloupností bitů na délky bitových sekvencí komplexního Huffmanova kódu

Posloupnost bitů	**Délka**
00	0
01	3
10	4
110	2
1110	1
1111	5

Zde je použit výše zmíněný parametr s možnými hodnotami 0, 2, 3. Ten totiž značí začátek procházení abecedy speciálních kódů – pokud je např. parametr 3, pak přeskočíme první tři speciální kódy (1, 2, 3) a začneme až od speciálního kódu 4. Po přiřazení délek jednotlivým speciálním kódům rekonstruujeme Huffmanův kód který generuje tyto speciální kódy, a ten použijeme podle postupu výše.

Co nám zde stále ještě chybí, je vědět kdy přestat procházet symboly, jak cílové abecedy tak abecedy se speciálními kódy. Jedna možnost je, samozřejmě, přestat když se dostaneme na konec abecedy. Druhá možnost je, přestat až je prostor binárního stromu konstruovaného Huffmanova kódu zaplněn.

Určeme si, že chceme aby ve stromu našeho Huffmanova kódu byla nejdelší možná větev (cesta od kořene k listu) délky 15, neboli sekvence 15 nul a jedniček. Naším prostorem tak bude 2^{15}, neboli 32768. Vždy, když přidáme do stromu novou větev, tak z tohoto prostoru odebereme 2^{15-x}, kde *x* je délka větve. Až zbývající prostor klesne na 0, pak už není dále možné přidat do stromu další větev.

Můžeme si to také ukázat na příkladu s bitovými sekvencemi (00, 01, 10, 110, 1110, 1111):

1. Začneme s prostorem 32768
2. Přidáme větev 00, délka 2, odebereme $2^{15-2} = 2^{13} = 8192$, zbývající prostor je 24576
3. Přidáme větev 01, délka 2, odebereme $2^{15-2} = 2^{13} = 8192$, zbývající prostor je 16384
4. Přidáme větev 10, délka 2, odebereme $2^{15-2} = 2^{13} = 8192$, zbývající prostor je 8192
5. Přidáme větev 110, délka 3, odebereme $2^{15-3} = 2^{12} = 4096$, zbývající prostor je 4096
6. Přidáme větev 1110, délka 4, odebereme $2^{15-4} = 2^{11} = 2048$, zbývající prostor je 2048
7. Přidáme větev 1111, délka 4, odebereme $2^{15-4} = 2^{11} = 2048$, zbývající prostor je 0

Povšimněme si, že pokud začneme s prázdným binárním stromem a přidáme větev délky 1, tj. přidáme např. symbol se sekvencí bitů 0, pak je celá levá polovina stromu využita, což se také projeví tím že prostor se zmenší přesně o polovinu. Pokud bychom namísto toho přidali větev délky 2, tj. například symbol se sekvenci bitů 00, pak se prostor zmenší o čtvrtinu, atd.

3.3. Parametry datového proudu

Hlavička s parametry proudu obsahuje pouze velikost posuvného okénka.

3.3.1. Posuvné okénko

Velikost posuvného okénka udává kolik posledně dekomprimovaných bajtů musíme uchovat v paměti, abychom se na ně mohli odkázat ve zpětných referencích.

Brotli podporuje posuvná okénka o velikostech přibližně 1 KiB až 16 MiB. Pro zápis velikosti je použit kód o délce 1, 4, nebo 7 bitů, podle následující tabulky. Pro jednodušší pochopení jsou posloupnosti bitů v tabulce odděleny po třech bitech, bity uvnitř každé skupiny čteme zprava doleva (tj. [1][110] je ve skutečnosti 1011):

Tabulka 4: Mapování posloupností bitů na velikost posuvného okénka

Posloupnost bitů	Velikost okénka
[1][000][001]	chybná hodnota
[1][000][010]	1008 (= 2^{10} - 16)

[1][000][011]	2032 (= 2^{11} - 16)
[1][000][100]	4080 (= 2^{12} - 16)
[1][000][101]	8176 (= 2^{13} - 16)
[1][000][110]	16368 (= 2^{14} - 16)
[1][000][111]	32752 (= 2^{15} - 16)
[0]	65520 (= 2^{16} - 16)
[1][000][000]	131056 (= 2^{17} - 16)
[1][001]	262128 (= 2^{18} - 16)
[1][010]	524272 (= 2^{19} - 16)
[1][011]	1048560 (= 2^{20} - 16)
[1][100]	2097136 (= 2^{21} - 16)
[1][101]	4194288 (= 2^{22} - 16)
[1][110]	8388592 (= 2^{23} - 16)
[1][111]	16777200 (= 2^{24} - 16)

3.4. Hlavička meta-bloku

Hlavička meta-bloku obsahuje informace potřebné k rekonstrukci jeho dat, a informace o existenci a případné pozici následujícího meta-bloku. Následující podkapitoly popisují jednotlivé části hlavičky, v pořadí v jakém se nacházejí v bitovém proudu.

3.4.1. Začátek hlavičky

Na začátku hlavičky je 1 bit který značí jestli je tento meta-blok poslední.
Pokud ano, následuje 1 bit který značí jestli je meta-blok prázdný.

Pokud je meta-blok poslední i prázdný, dekomprese je ukončena a všechny zbývající bity musí být 0.

3.4.2. Velikost dat

Velikost dat je počet dekomprimovaných bajtů které jsou přidány do výstupního proudu po přečtení tohoto meta-bloku. Definice začíná 2 bity které určí počet následujících půlbajtů (půlbajt, také nazýván *nibble*, je skupina 4 bitů) obsahujících přesnou velikost dat:

Tabulka 5: Mapování hodnot 2 bitů na počet půlbajtů tvořících velikost dat meta-bloku

Hodnota přečtených bitů	0	1	2	3
Počet půlbajtů	4	5	6	0

Nastane jeden ze dvou případů:

1. **Počet půlbajtů je 0**, meta-blok je tedy prázdný, a dále obsahuje jen informaci o tom kolik bajtů musíme přeskočit abychom se dostali k dalšímu meta-bloku

- Následuje 1 rezervovaný nulový bit
- Přečteme 2 bity které určí počet následujících bajtů, které obsahují informaci kolik bajtů musíme přeskočit
  - Pokud je tento počet 0, pak počet bajtů k přeskočení je také 0
  - Pokud tento počet není 0, přečteme (8 × počet bajtů) bitů ze vstupního proudu a k přečtené hodnotě přičteme 1, což nám dá počet bajtů k přeskočení
- Přejdeme na čtení po znacích, takže musíme zarovnat pozici v proudu bitů na násobek 8, a poté provedeme přeskočení daného počtu bajtů
- Následuje hlavička dalšího meta-bloku

2. **Počet půlbajtů je větší než 0**
   - Pro získání přesné velikosti dat přečteme (4 × počet půlbajtů) bitů ze vstupního proudu, a k přečtené hodnotě přičteme 1, což každému meta-bloku dává kapacitu až 16 MiB
   - Pokud meta-blok není prázdný, následuje 1 bit který značí zda obsahuje nekomprimovaná data, a pokud ano tak ukončíme čtení hlavičky a přejdeme ke čtení dat meta-bloku

Prázdné meta-bloky umožňují přidání vlastních metadat do komprimovaného datového proud Brotli bez narušení kompatibility s existujícími dekompresory. Tuto možnost mohou využít upravené verze kompresoru a dekompresoru pro přidání vlastní funkcionality kterou Brotli postrádá, například podporu kontrolního součtu kterým se kontroluje zda přenos proběhl v pořádku, nebo informace o pozicích a počátečních stavech meta-bloků pro umožnění paralelní dekomprese.

3.4.3. Informace o kategoriích a skupinách kódů

Každá kategorie kódu příkazu *insert&copy* (literály, insert&copy délky, vzdálenosti) může mít až 256 skupin s různými Huffmanovými kódy. Při čtení datové části má každá kategorie daný přesný počet kódů které zbývají v aktuální skupině dané kategorie, a když je zbývající počet 0 tak je přečten příkaz *block-switch* který určí novou skupinu a počet kódů v ní.

Následující proces provedeme pro všechny 3 kategorie:

1. Přečteme kód s proměnnou délkou 1-11 bitů, který určuje celkový počet skupin
2. Pokud je počet skupin roven 1, nastavíme počet zbývajících kódů ve skupině na 2^{24}
3. Pokud je počet skupin větší než 1, pak:
   1. Přečteme Huffmanův kód který je použit pro čtení následující skupiny; abeceda tohoto kódu obsahuje indexy všech skupin plus 2 speciální hodnoty které budou popsány v kapitole čtení komprimovaných dat
   2. Přečteme Huffmanův kód který je použit pro čtení počtu zbývajících kódů; abeceda tohoto kódu obsahuje 26 hodnot popsaných níže

3. Nastavíme index první skupiny na 0
4. Přečteme počet zbývajících kódů v první skupině

Abeceda kódu pro čtení počtu zbývajících kódů obsahuje 26 hodnot které reprezentují speciální kódy. Každý z těchto kódů určuje kolik bitů musíme dále přečíst ze vstupního proudu, a počáteční hodnotu rozmezí ke kterému je hodnota přečtených bitů přičtena:

Tabulka 6: Reference pro převedení speciálního kódu na počet zbývajících kódů skupiny kategorie

Kód	Počet bitů	Rozmezí hodnot	Kód	Počet bitů	Rozmezí hodnot
0	2	[1, 4]	13	5	[145, 176]
1		[5, 8]	14		[177, 208]
2		[9, 12]	15		[209, 240]
3		[13, 16]	16	6	[241, 304]
4	3	[17, 24]	17		[305, 368]
5		[25, 32]	18	7	[369, 496]
6		[33, 40]	19	8	[497, 752]
7		[41, 48]	20	9	[753, 1 264]
8	4	[49, 64]	21	10	[1 265, 2 288]
9		[65, 80]	22	11	[2 289, 4 336]
10		[81, 96]	23	12	[4 337, 8 432]
11		[97, 112]	24	13	[8 433, 16 624]
12	5	[113, 114]	25	24	[16 625, 16 793 840]

3.4.4. Parametry vzdáleností

Následují dva parametry které upravují výpočet vzdáleností, NPOSTFIX a NDIRECT:

- NPOSTFIX je hodnota v rozmezí 0-3, získaná přečtením 2 bitů
- NDIRECT je hodnota v rozmezí 0-120, získaná přečtením 4 bitů a posunutím těchto bitů doleva (*bitwise left-shift*) o NPOSTFIX bitů

Pro kompresi fontů ve formátu WOFF2 jsou použity napevno nastavené hodnoty NPOSTFIX = 1, NDIRECT = 12. U nejvyšších úrovní komprese jsou zkoušeny různé hodnoty obou parametrů.

3.4.5. Kontextové modelování

Kontextové modelování se skládá ze seznamu módů pro literály, kontextové mapy pro literály, a kontextové mapy pro vzdálenosti v tomto pořadí.

Každá skupina kategorie literálů má přiřazen kontextový mód. Každý kontextový mód je zapsán 2 bity:

Tabulka 7: Mapování hodnot 2 bitů na kontextový mód pro literály

Hodnota přečtených bitů	0	1	2	3
Kontextový mód	LSB6	MSB6	UTF8	Signed

Následují dvě kontextové mapy, jedna pro literály (64 hodnot na skupinu) a jedna pro vzdálenosti (4 hodnoty na skupinu).

Přečteme kód s proměnnou délkou 1-11 bitů který určuje celkový počet Huffmanových kódů v dané kategorii. Pokud je tento počet 1, pak kontextová mapa obsahuje pouze nuly (tzn. všechny hodnoty odkazují na první a jediný Huffmanův kód). Pokud je tento počet 2 nebo více, následuje konkrétní definice kontextové mapy.

Kontextová mapa je postupně naplněna indexy odkazujícími na Huffmanovy kódy dané kategorie. Připomeňme si, že velikost kontextové mapy je (*počet skupin* × *počet kódů na skupinu*), s tím že můžeme mít až 256 skupin, ať už literálů nebo vzdáleností. Nejvyšší potenciální velikost kontextové mapy je tedy (256 × 64) = 16 384 prvků, Brotli proto používá několik optimalizací:

1. Může být použit run-length encoding, který je vhodný pro zakódování dlouhých sekvencí (v tomto případě jsou takto zakódovány dlouhé sekvence nul)
2. Hodnoty kontextové mapy i run-length kódů jsou zakódovány Huffmanovým kódem
3. Po naplnění kontextové mapy může být provedena tzv. *inverse move-to-front* transformace

Začneme přečtením jednoho bitu, který určuje zda je použit run-length encoding. Pokud ano, pak následují 4 bity jejichž hodnota určuje počet run-length kódů.

Dále přečteme definici Huffmanova kódu. Při naplňování kontextové mapy vždy přečteme kód jehož hodnotu označíme *x*:

- Pokud je *x* rovno 0, pak do mapy vložíme 0
- Pokud platí (1 <= x <= počet run-length kódů), pak:
  - Přečteme dalších *x* bitů ze vstupního proudu, jejichž hodnotu označíme *y*
  - Vložíme do mapy (2^x + y - 1) nul
- Pokud platí (x > počet run-length kódů), pak:
  - Vložíme do mapy hodnotu x od kterého odečteme počet run-length kódů

Nakonec přečteme jeden bit který určuje zda použít *inverse move-to-front* transformaci, dále IMTF. Jedná se o inverzní funkci k funkci *move-to-front*, která je použita například jako součást kompresního algoritmu bzip2, a funguje tak, že namísto kódování znaku dané abecedy kóduje jen jeho index v abecedě, a poté vždy tento znak přesune na začátek abecedy.

Funkce IMTF je vhodná pro případ kdy chceme kompaktně zakódovat dlouhé sekvence čísel, protože způsob jakým je zde použit run-length encoding umožňuje zakódovat dlouhé sekvence nul které generuje funkce *move-to-front*. Uveďme si příklad:

1. Abeceda se skládá z posloupnosti čísel v intervalu [0, 255], zahrnuje tedy všechny možné hodnoty v bajtu
2. Chceme zakódovat řetězec: (0, 0, 1, 1, 1, 1, 2, 2, 2, 0, 0)
3. Provedeme funkci *move-to-front*:

Tabulka 8: Ukázka postupu zakódování řetězce s danou abecedou funkcí move-to-front

Originální řetězec	Zakódovaný řetězec	Abeceda
0	0	0,1,2,3,4...
0,0	0,0	0,1,2,3,4...
0,0,1	0,0,1	1,0,2,3,4...
0,0,1,1	0,0,1,0	1,0,2,3,4...
0,0,1,1,1	0,0,1,0,0	1,0,2,3,4...
0,0,1,1,1,1	0,0,1,0,0,0	1,0,2,3,4...
0,0,1,1,1,1,2	0,0,1,0,0,0,2	2,1,0,3,4...
0,0,1,1,1,1,2,2	0,0,1,0,0,0,2,0	2,1,0,3,4...
0,0,1,1,1,1,2,2,2	0,0,1,0,0,0,2,0,0	2,1,0,3,4...
0,0,1,1,1,1,2,2,2,0	0,0,1,0,0,0,2,0,0,2	0,2,1,3,4...
0,0,1,1,1,1,2,2,2,0,0	0,0,1,0,0,0,2,0,0,2,0	0,2,1,3,4...

V zakódovaném řetězci se nachází mnohem více nul, které Brotli umí při kompresi kompaktně uložit, a při dekompresi je po přečtení takto zakódovaný řetězec dekódován funkcí IMTF.

3.4.6. Huffmanovy kódy kategorií

Hlavička je zakončena definicí všech Huffmanových kódů pro literály, insert&copy délky, a vzdálenosti v tomto pořadí.

Počet kódů pro literály a vzdálenosti je určen při čtení příslušné kontextové mapy, počet kódů pro insert&copy délky je roven počtu skupin této kategorie. Při čtení každého kódu potřebujeme znát velikost abecedy čtené kategorie:

- **Literály**: 256 (8 bitů)
- **I&C délky**: 704 (10 bitů)
- **Vzdálenosti**: $16 + \text{NDIRECT} + 48 \times 2^{\text{NPOSTFIX}}$ (proměnný počet bitů)

3.5. Data meta-bloku (nekomprimovaná)

Nekomprimovaná data jsou uložena jako sekvence čistých bajtů. Pro jejich přečtení nejdříve přejdeme na čtení po znacích, takže musíme zarovnat pozici ve vstupním proudu bitů na násobek 8. Jelikož hlavička definuje velikost dat v bajtech, tak přečteme přesně tolik bajtů kolik určuje hlavička a uložíme je do výstupního proudu. Poté následuje hlavička dalšího meta-bloku.

3.6. Data meta-bloku (komprimovaná)

Komprimovaná datová část meta-bloku obsahuje sekvenci *insert&copy* příkazů. Nezapomeňme, že jsou zde 3 kategorie kódů jejichž hodnoty jsou zakódovány Huffmanovými kódy, a před každým přečtením každého kódu musíme zkontrolovat zda je potřeba aktualizovat skupinu dané kategorie, kterou pak provede příkaz *block-switch*.

Pro dekódování literálů potřebujeme uchovat poslední 2 dekomprimované bajty, které slouží jako parametry kontextového modelu. K tomu použijeme cyklickou frontu s 2 prvky, kterou aktualizujeme po každém zapsání do výstupního proudu (tj. po přečtení každého literálu a také po přečtení zpětné reference). Frontu inicializujeme s hodnotami (0, 0) na začátku každého meta-bloku.

Pro dekódování kódu vzdáleností potřebujeme uchovat poslední 4 hodnoty vzdáleností, k čemuž opět použijeme cyklickou frontu, inicializovanou hodnotami (16, 15, 11, 4). Inicializace proběhne pouze začátku dekomprese celého proudu – fronta hodnot vzdáleností je součást sdíleného stavu, ke kterému přistupují všechny meta-bloky. Parametrem kontextového modelu vzdáleností je hodnota *copy length*.

Algoritmus postupuje takto:

1. Přečteme kód insert&copy délek, a dekódujeme jej podle postupu níže
    - Po dekódování získáme tři parametry: *insert length, copy length, distance code zero*
2. Přečteme *insert length* literálů (bajtů), každý a uložíme je do výstupního proudu
3. Pokud je počet dosud přečtených bajtů roven velikosti dat meta-bloku, čtení dat je přerušeno
4. Nyní potřebujeme zjistit vzdálenost, zde nastane jeden ze dvou případů:
    - Pokud je pravdivostní hodnota *distance code zero* pravdivá, pak je kód vzdálenosti nastaven na nulu, což je speciální hodnota která zopakuje posledně použitou vzdálenost uloženou v cyklické frontě
    - Pokud je nepravdivá, přečteme kód vzdálenosti a dekódujeme jej, což vysvětlí jedna z následujících podkapitol
5. Jelikož vzdálenost může odkazovat buď na zpětnou referenci nebo referenci na statický slovník, zde také nastane jeden ze dvou případů:
    - Pokud je vzdálenost menší nebo rovna počtu dosud dekomprimovaných bajtů ze všech meta-bloků, ale není větší než velikost posuvného okénka, pak se jedná o zpětnou referenci která zkopíruje *copy length* dříve zapsaných dekomprimovaných bajtů

    - Pokud je vzdálenost větší, pak se jedná o referenci na slovo délky *copy length* ve statickém slovníku, kterou dekódujeme podle postupu níže
6. Opět zkontrolujeme zda je počet dosud přečtených bajtů roven velikosti meta-bloku, pokud ano tak ukončíme čtení dat, pokud ne tak se vracíme ke kroku 1

Pokud je hlavička označena jako poslední, pak je dekomprese ukončena (zbývající bity v posledním bajtu jsou nulové), v opačném případě následuje další meta-blok.

3.6.1. Dekódování příkazu block-switch

Pokud potřebujeme aktualizovat skupinu nějaké kategorie, musíme nastavit novou skupinu a počet zbývajících kódů. U každé kategorie si musíme pamatovat předposlední použitou skupinu, na začátku ji inicializujeme na hodnotu 1.

Použijeme Huffmanův kód pro přečtení kódu následující skupiny:

- Pokud je kód roven 0, pak zopakujeme předposlední skupinu
- Pokud je kód roven 1, pak k aktuální skupině přičteme jedničku; pokud je nová hodnota větší než počet skupin, tak od ní počet skupin odečteme abychom se vrátili na hodnotu 0
- Pokud je kód větší než 1, pak je novou skupinou (*kód* - 2)

Pak přečteme počet zbývajících kódů stejně jako jsme to udělali při čtení hlavičky. Tímto příkaz block-switch končí a můžeme pokračovat se čtením kódu kategorie.

3.6.2. Dekódování kódu insert&copy délek

Kód insert&copy délek má 704 možných hodnot. Pro jeho přečtení je použit jediný možný Huffmanův kód aktuální skupiny, přečtenou hodnotu dekódujeme s pomocí tabulky:

Tabulka 9: Reference pro extrakci páru insert a copy kódů z hodnoty insert&copy délek

	Copy kód		
Insert kód	**0-7**	**8-15**	**16-23**
0-7	[0, 63] [128, 191]	[64, 127] [192, 255]	[384, 447]
8-15	[256, 319]	[320, 383]	[512, 575]
16-23	[448, 511]	[576, 639]	[640, 703]

Podle tabulky hned určíme pravdivostní hodnotu *distance code zero*, která je pravdivá pokud je kód v rozmezí 0-127, neboli v jedné z prvních dvou buněk tabulky.

Kód insert&copy délek si můžeme představit jako sekvenci bitů: `MSB [gggg][iii][ccc] LSB`

- *gggg* jsou 4 bity určující pozici v mřížce podle které zjistíme rozmezí *insert kódu* a *copy kódu*
- *iii* jsou 3 bity určující posunutí uvnitř rozmezí *insert kódu*

- *ccc* jsou 3 bity určující posunutí uvnitř rozmezí *copy kódu*

Každá hodnota *insert kódu* a *copy kódu* určuje kolik bitů musíme dále přečíst ze vstupního proudu, a počáteční hodnotu rozmezí ke kterému je hodnota přečtených bitů přičtena, abychom získali finální hodnoty *insert length* a *copy length*.

Tabulky insert kódů a copy kódů

Tabulka 10: Reference pro převedení insert kódu na hodnotu insert length

Kód	Počet bitů	Rozmezí hodnot	Kód	Počet bitů	Rozmezí hodnot
0	0	0	12	4	[34, 49]
1		1	13		[50, 65]
2		2	14	5	[66, 97]
3		3	15		[98, 129]
4		4	16	6	[130, 193]
5		5	17	7	[194, 321]
6	1	[6, 7]	18	8	[322, 577]
7		[8, 9]	19	9	[578, 1 089]
8	2	[10, 13]	20	10	[1 090, 2 113]
9		[14, 17]	21	12	[2 114, 6 209]
10	3	[18, 25]	22	14	[6 210, 22 593]
11		[26, 33]	23	24	[22 594, 16 799 809]

Tabulka 11: Reference pro převedení copy kódu na hodnotu copy length

Kód	Počet bitů	Rozmezí hodnot	Kód	Počet bitů	Rozmezí hodnot
0	0	2	12	3	[22, 29]
1		3	13		[30, 37]
2		4	14	4	[38, 53]
3		5	15		[54, 69]
4		6	16	5	[70, 101]
5		7	17		[102, 133]
6		8	18	6	[134, 197]
7		9	19	7	[198, 325]
8	1	[10, 11]	20	8	[326, 581]
9		[12, 13]	21	9	[582, 1 093]
10	2	[14, 17]	22	10	[1 094, 2 177]
11		[18, 21]	23	24	[2 118, 16 779 333]

3.6.3. Dekódování kódu vzdálenosti

Pro přečtení kódu vzdálenosti použijeme jeden ze 4 možných Huffmanových kódů aktuální skupiny, který z nich je použit závisí na hodnotě *copy length* jak bylo popsáno v kapitole kontextového modelování. Také potřebujeme parametry NPOSTFIX a NDIRECT které jsou definovány v hlavičce, a 4 posledně použité hodnoty vzdáleností.

U kódu vzdálenosti nastane jeden ze tří případů:

- Pokud platí (kód < 16), pak se jedná o speciální kód který odkazuje na jednu z předchozích vzdáleností uložených v cyklické frontě:

  Tabulka 12: Významy speciálních kódů odkazujících na předchozí použité vzdálenosti

Kód	Hodnota	Kód	Hodnota
0	poslední hodnota	8	poslední hodnota - 3
1	předposlední hodnota	9	poslední hodnota + 3
2	třetí hodnota od konce	10	předposlední hodnota - 1
3	čtvrtá hodnota od konce	11	předposlední hodnota + 1
4	poslední hodnota - 1	12	předposlední hodnota - 2
5	poslední hodnota + 1	13	předposlední hodnota + 2
6	poslední hodnota - 2	14	předposlední hodnota - 3
7	poslední hodnota + 2	15	předposlední hodnota + 3

- Pokud platí (16 <= kód < 16 + NDIRECT), pak je hodnota vzdálenosti rovna (kód - 15)
- Pokud platí (16 + NDIRECT <= kód), pak je proveden algoritmus pro výpočet hodnoty

Brotli definuje algoritmus pro výpočet hodnoty vzdálenosti. Definujme *dcode* jako kód vzdálenosti od kterého odečteme (16 + NDIRECT), a *dvalue* jako hodnotu vzdálenosti kterou chceme vypočítat. Budeme také potřebovat přečíst několik dalších bitů ze vstupního proudu.

- *dcode* je zkonstruován ze sekvence bitů: `MSB [extracount][x][postfix] LSB`
- *dvalue* je zkonstruován ze sekvence bitů: `MSB [1][x][extravalue][postfix] LSB`

Vysvětlení jednotlivých částí:

- *postfix* jsou 0-3 spodní bity hodnoty *dcode* (počet je dán parametrem NPOSTFIX)
- *x* je jeden bit v hodnotě *dcode* který následuje za *postfix* bity
- *extracount* je počet bitů které musíme přečíst ze vstupního proudu
- *extravalue* je hodnota těchto přečtených bitů
- na stranu nejvýznamnějšího bitu hodnoty *dvalue* je vždy přidána jednička

Abychom získali hodnotu *dvalue*, tak musíme nejdříve provést několik bitových operací abychom extrahovali jednotlivé komponenty hodnoty *dcode*, přečíst *extracount* bitů ze vstupního proudu, a pak opět pomocí bitových operací poskládat komponenty tvořící hodnotu *dvalue* do jedné hodnoty. Specifikace Brotli obsahuje pseudokód s optimalizovanými výpočty které provádí tyto operace.

Finální hodnotou vzdálenosti je potom (dvalue + 1 + NDIRECT), to proto, že všechny hodnoty vzdálenosti od 1 po NDIRECT pokryje předchozí případ (16 <= kód < 16 + NDIRECT).

3.6.4. Dekódování reference na statický slovník

Slova jsou ve slovníkovém souboru seřazena podle velikosti, od 4 do 24 bajtů na slovo. Dále formát definuje počet slov každé velikosti.

Tabulka 13: Počty slov ve statickém slovníku podle jejich velikosti

Velikost	Počet slov	Velikost	Počet slov	Velikost	Počet slov
4	$1024 = 2^{10}$	11	$1024 = 2^{10}$	18	$256 = 2^{8}$
5	$1024 = 2^{10}$	12	$1024 = 2^{10}$	19	$128 = 2^{7}$
6	$2048 = 2^{11}$	13	$512 = 2^{9}$	20	$128 = 2^{7}$
7	$2048 = 2^{11}$	14	$512 = 2^{9}$	21	$64 = 2^{6}$
8	$1024 = 2^{10}$	15	$256 = 2^{8}$	22	$64 = 2^{6}$
9	$1024 = 2^{10}$	16	$128 = 2^{7}$	23	$32 = 2^{5}$
10	$1024 = 2^{10}$	17	$128 = 2^{7}$	24	$32 = 2^{5}$

Velikost slova je dána hodnotou *copy length*. Indexy slova a transformace jsou zakódovány ve vzdálenosti. Před dalšími výpočty nesmíme zapomenout od vzdálenosti odečíst hranici kde začínají reference na slovník, aby při výpočtech vzdálenost 0 vždy odkazovala na první slovo.

- Index slova je: *vzdálenost* modulo *počet slov velikosti*
- Index transformace je: *vzdálenost* děleno *počet slov velikosti* (dělení beze zbytku)

Počty slov jsou mocninami dvou, což umožňuje použít optimalizaci v kódu kde místo dělení počtem slov stačí posunout bity doprava (*bitwise right-shift*), např. posunutí o 5 bitů doprava je stejné jako bezezbytkové dělení 2^5.

Když známe index slova, najdeme jeho pozici v souboru slovníku spočítáním kolik bajtů zabírají všechna předcházející slova, přečteme jej, a aplikujeme transformaci. Seznam všech transformací je uveden v příloze.

4. Testování Brotli a dalších algoritmů

První část praktické práce byla implementace dekomprese v konzolové aplikaci napsané v jazyce C#. Ve druhé části budeme testovat účinnost a rychlost komprese a dekomprese Brotli, a porovnávat jej s několika příbuznými algoritmy. Při testování a porovnávání algoritmů se zaměříme na tyto vlastnosti, vždy pro nejnižší a nejvyšší úroveň komprese:

- Kompresní poměr ($\frac{nekomprimovaná\,velikost}{komprimovaná\,velikost}$)
- Úspora místa ($1-\frac{komprimovaná\,velikost}{nekomprimovaná\,velikost}$) značená v procentech
- Rychlost komprese ($\frac{nekomprimovaná\,velikost}{doba\,komprese}$) v Mbit/s
- Rychlost dekomprese ($\frac{nekomprimovaná\,velikost}{doba\,dekomprese}$) v Mbit/s

U Brotli budeme testovat všechny kompresní úrovně. Při porovnávání jednotlivých úrovní také budeme sledovat statistiky o využití součástí Brotli:

- Počty meta-bloků a jejich typů (komprimovaný, nekomprimovaný, prázdný)
- Počty slov a využití transformačních funkcí statického slovníku
- Počty skupin a Huffmanových kódů jednotlivých kategorií
- Počty kontextových módů pro literály

Jelikož je Brotli primárně zaměřen na kompresi webových zdrojů, podíváme se také na statistiky o aktuální podpoře prohlížečů a nejnavštěvovanějších webů.

Pro zajímavost také uvedeme statistiky z porovnání komprese formátů fontů WOFF 1.0 a WOFF 2.0. Jelikož se jedná o binární formáty jejichž části obsahují komprimovaná data, konkrétně DEFLATE pro verzi 1.0 a Brotli pro verzi 2.0, a používají specializované kompresory a dekompresory které jsou mimo rámec této práce, porovnáme pouze velikosti souborů fontů v obou formátech.

4.1. Testované algoritmy a úrovně komprese

Všechny testované algoritmy jsou založeny na LZ77. Algoritmy byly vybrány na základě historické významnosti s Brotli (DEFLATE, Zopfli), a zajímavých vlastností (efektivita LZMA2, rychlost LZ4).

4.1.1. DEFLATE

DEFLATE je populární kompresní algoritmus který kombinuje LZ77 a Huffmanovo kódování.[6] Využívá jej řada formátů souborů, ať už archivní formáty jako je např. ZIP, obrázkový formát PNG, nebo první verze formátu fontů WOFF.

Novější verze .NETu obsahují implementaci DEFLATE pomocí knihovny zlib. Budeme testovat nejnižší úroveň 1 a nejvyšší úroveň 9, k tomu ovšem musíme využít systém *reflexe* pro modifikaci skrytého interního stavu objektů, protože veřejné API .NETu umožňuje nastavit pouze úrovně 1 a 6.

4.1.2. LZMA2

LZMA je algoritmus vytvořen pro známý archivní formát 7z.[7] LZMA2 je pak vylepšením formátu i algoritmu, který podporuje kompresi s využitím více vláken procesoru, a kombinování více proudů nekomprimovaných dat a komprimovaných LZMA dat najednou.

Při testování použijeme knihovnu pro kompresi a dekompresi souborů ve formátu XZ[8], který obsahuje LZMA2 data. Komprese bude limitovaná jedním vláknem, a budeme testovat nejnižší úroveň 0, a nejvyšší úroveň 9.

4.1.3. LZ4

LZ4 se zaměřuje na velmi vysokou rychlost komprese i dekomprese. Podle oficiálních testů, které algoritmus porovnávají například s LZO, starším algoritmem který se také zaměřil na rychlost, je jeho kompresní poměr srovnatelný s algoritmem LZO, ale dekomprese je až 3-4x rychlejší.[9]

LZ4 má dva módy které budou zahrnuty v testech:

- Základní mód má vysokou rychlost komprese i dekomprese na úkor kompresního poměru
- *High Compression* mód komprimuje mnohem pomaleji ale s vyšším kompresním poměrem, rychlost dekomprese je však stejná nebo rychlejší než u základního módu

4.1.4. Zopfli

Jedná se o variantu DEFLATE zaměřující se na co nejvyšší účinnost komprese na úkor rychlosti.[10] Algoritmus byl vytvořen Googlem a stal se předchůdcem Brotli.

Zopfli je specifikován pouze jako kompresní algoritmus, jeho výstupem je datový proud DEFLATE, případně jeden z konkrétních formátů obsahujících DEFLATE data – gzip, zlib, nebo PNG.

Pro dekompresi je možné použít jakýkoliv existující algoritmus který umí pracovat s DEFLATE nebo se zmíněnými formáty. Tato zpětná kompatibilita činí Zopfli vhodným algoritmem pro kompresi statických webových zdrojů, ať už mluvíme o textových souborech (např. HTML, CSS, JS), obrázcích

6 DEUTSCH, Peter. RFC 1951. *IETF Tools* [online]. 1996 [cit. 2018-04-12]. Dostupné z: https://tools.ietf.org/html/rfc1951
7 7z Format. *7-Zip* [online]. c2018 [cit. 2018-04-12]. Dostupné z: https://www.7-zip.org/7z.html
8 The .xz File Format. *The Tukaani Project* [online]. [cit. 2009-08-27]. Dostupné z: https://tukaani.org/xz/xz-file-format.txt
9 *LZ4* [online]. [cit. 2018-04-12]. Dostupné z: http://lz4.github.io/lz4
10 Zopfli. *GitHub* [online]. [cit. 2017-07-07]. Dostupné z: https://github.com/google/zopfli

ve formátu PNG, nebo fontech ve formátu WOFF, ale kvůli jeho velmi nízké rychlosti není vhodný pro dynamicky generovaný obsah.

Zopfli nemá klasické nastavení úrovně komprese, obsahuje jen několik parametrů které lze v kompresoru nastavovat individuálně. Hlavním parametrem je počet iterací který nejvíce ovlivňuje výslednou velikost a dobu komprese. V rámci testování použijeme výchozí hodnoty všech parametrů s počtem iterací 15, což je hodnota doporučena pro menší soubory.

4.1.5. Brotli

Hlavním cílem testování je samozřejmě Brotli.

Velikost posuvného okénka může být nastavena manuálně při kompresi, pokud jej nijak nenastavíme tak je použito 22bitové posuvné okénko (4 194 288 bajtů) které použijeme v testech.

Brotli má 12 úrovní komprese. Některé úrovně upravují parametry komprese, některé úrovně kompletně vypínají součásti formátu.[11] V kompresoru je také definovaná řada různých algoritmů s různými parametry pro generování zpětných referencí a referencí na slovník, který algoritmus a které jeho parametry jsou použity závisí na úrovni komprese, velikosti posuvného okénka, a velikosti vstupních dat.

- **Úrovně 0-1** používají velmi jednoduchý a rychlý algoritmus, nejsou mezi nimi velké rozdíly
- **Úrovně 2-3** lépe vyhledávají zpětné reference což vede k mnohem lepší kompresi než u předchozích úrovní, ale nevyužívají některé pokročilé součásti Brotli
- **Úroveň 4** je první úrovní která umí rozdělovat kategorie kódů příkazu *insert&copy* v rámci jednoho meta-bloku do několika skupin
- **Úrovně 5-9** opět mění algoritmus vyhledávání zpětných referencí a začínají používat kontextové modelování literálů, se zvyšující úrovní se postupně zvyšuje i efektivita jednotlivých částí
- **Úrovně 10-11** využívají v algoritmu vyhledávání zpětných referencí principy Zopfli, jsou mnohem efektivnější, ale také několikanásobně pomalejší než předchozí úrovně; kompresor zde mimo jiné testuje různé hodnoty parametrů vzdáleností, a začíná používat kontextové modelování vzdáleností

4.2. Metodika testování

Všechny testy a statistiky jsou také provedeny konzolovou aplikací vytvořené v praktické části práce, napsanou v C# a cílenou na .NET Framework 4.5.2.

Pro co nejspravedlivější porovnávání jednotlivých algoritmů použijeme jejich nativní knihovny, které lze volat pomocí *Platform Invoke* funkcionality v .NETu. Toto propojení s využitím *Platform Invoke* zajišťují tzv. *language binding* knihovny.

11 Brotli. *GitHub* [online]. [cit. 2018-04-13]. Dostupné z: https://github.com/google/brotli/blob/68db5c/c/enc/quality.h

- **DEFLATE**
  - Počínaje verzí 4.5 .NETu je v implementaci DEFLATE použita nativní knihovna zlib
- **LZMA2**
  - Binding knihovna *ManagedXZ 1.0.0*[12]
  - Nativní knihovna *lzma 5.2.1*[13]
- **LZ4**
  - Binding knihovna *Lz4.Net 1.0.98*[14]
  - Nativní knihovna *lz4 r94*[15]
- **Zopfli**
  - Zopfli je pouze kompresní algoritmus, pro dekompresi byl použit DEFLATE v .NETu
  - Kvůli kritickým chybám bylo použito vlastní sestavení knihovny *zopfli* z oficiálního zdrojového kódu
  - Binding knihovna *libzopfli-sharp 1.0.14325.1*[16]
  - Nativní knihovna *zopfli master@64c6f36*[17]
- **Brotli**
  - Vlastní implementace dekompresoru je použita pro získání detailních statistik o každém souboru, nativní implementace je použita pro testování výkonu
  - Binding knihovna *System.IO.Compression.Brotli 0.1.0-e171202-1*[18]
  - Nativní knihovna *brotli 1.0.4*[19]

Jako testovací systém byl použit Windows 10 s .NET Frameworkem 4.7.2, procesorem AMD Ryzen 1200 @ 3.10 GHz, a pamětí RAM o frekvenci 2 666 MHz v single-channel módu. Aplikace byla sestavena pro platformu x64.

Každý algoritmus měl k dispozici 1 vlákno procesoru, a všechna komprese a dekomprese proběhla v paměti RAM aby se eliminoval vliv zápisu a čtení z disku.

12 https://www.nuget.org/packages/ManagedXZ
13 https://tukaani.org/xz/
14 https://www.nuget.org/packages/Lz4.Net
15 https://github.com/lz4/lz4/releases
16 https://www.nuget.org/packages/libzopfli-sharp
17 https://github.com/google/zopfli/commit/64c6f362fefd56dccbf31906fdb3e31f6a6faf80
18 https://dotnet.myget.org/feed/dotnet-corefxlab/package/nuget/System.IO.Compression.Brotli
19 https://github.com/google/brotli/releases

Kvůli obrovským rozdílům v rychlostech algoritmů byly použity různé počty vzorků, ve výpočtech je pak doba komprese/dekomprese medián ze všech zaznamenaných časů.

- 100 vzorků pro LZ4
- 50 vzorků pro DEFLATE
- 25 vzorků pro LZMA2 a Brotli (úrovně 0-9)
- 10 vzorků pro Brotli (úrovně 10-11)
- 5 vzorků pro Zopfli (pouze komprese)

4.2.1. Testovací data

Protože se Brotli zaměřuje především na kompresi webového obsahu, velká část testovaných dat byla vybrána tak, aby reflektovala typické použití algoritmu v reálném světě. Jakožto jediný testovaný algoritmus který disponuje statickým slovníkem tak má Brotli u těchto dat oproti ostatním algoritmům výhodu. Celkem bylo otestováno 116 souborů s velikostmi 3,6 KiB až 9,72 MiB (průměr 564,9 KiB, medián 182,6 KiB).

The Canterbury Corpus

Korpus Canterbury je známým souborem dat sloužícím k testování algoritmů pro kompresi textu, nahrazující starší korpus Calgary.[20] Kromě hlavního Canterbury korpusu jsou k dispozici i další menší korpusy zaměřující se na specifické druhy dat[21], z nichž byly použity:

- The Large Corpus (E.coli, bible.txt, world192.txt)
- The Miscellaneous Corpus (pi.txt)

Silesia Corpus

Korpus Silesia je další známou kolekcí dat pro testování bezztrátové komprese, oproti korpusu Canterbury se zaměřuje na soubory o velikostech alespoň 5 MiB, a na kombinaci různých typů dat včetně obrázků, databází, a dalších binárních dat.

Jelikož testujeme především kompresi textu, použijeme z korpusu pouze dokumenty (dickens.txt, reymont.pdf) a namátkový výběr XML souborů o různých velikostech (elts.xml, stats1.xml, tal2.xml).

Projekt Gutenberg

Projekt Gutenberg je digitální knihovna obsahující literární díla volně k dispozici ke stažení v různých formátech.[22]

Z Projektu Gutenberg byla použita díla ve většině jazyků které zahrnuje statický slovník Brotli (angličtina, arabština, čínština, španělština, ruština), a také z francouzštiny která ve slovníku obsažena

20 Purpose of the Canterbury Corpus. *The Canterbury Corpus* [online]. [cit. 2000-12-11]. Dostupné z: http://corpus.canterbury.ac.nz/purpose.html

21 Descriptions of the corpora. *The Canterbury Corpus* [online]. [cit. 2001-01-08]. Dostupné z: http://corpus.canterbury.ac.nz/descriptions/

22 *Project Gutenberg* [online]. [cit. 2018-04-10]. Dostupné z: https://www.gutenberg.org

není. U každého jazyka bylo namátkově bylo vybráno alespoň 5 děl, vždy v textovém formátu s kódováním UTF-8. U anglických děl byly kromě čistého textu v testu zahrnuty také soubory ve formátu PDF, RTF, a HTML. Projekt Gutenberg obsahuje i numerická data, například seznam prvních 100 000 prvočísel který byl v testu také použit.

Při výběru byla upřednostněna díla z 20. a 21. století, u některých jazyků byly kvůli nedostatku novějších děl použity další zdroje:

- Pro arabštinu byl použit zdroj Ghazali[23], použitá díla byla ručně převedena do textového formátu
- Pro čínštinu byl použit seznam čínských slov pd-phrases[24], převeden do kódování UTF-8

Webové zdroje

Pro reprezentaci typických webových dat byl použit soubor známých CSS a JS knihoven, každá jak v minifikované tak neminifikované verzi, a jen jeden rozsáhlý HTML soubor navíc jelikož několik HTML a XML souborů je již zahrnuto v použitých korpusech.

- **CSS** – Animate 3.6.0, Bootstrap 3.3.7 & 4.0.0, Fontawesome Free 5.0.9, Pure 1.0.0, Semantic UI 2.3
- **JS** – Bootstrap 3.3.7 & 4.0.0, Fontawesome Free 5.0.9, Chart 2.7.2, jQuery 3.3.1, Moment 2.22.0, React 16.3.1, Redux 3.7.2, Semantic UI 2.3, Vue 2.5.16
- **HTML** – Large HTML page with Images (Borland)
- **WOFF** – Kolekce nejpoužívanějších fontů z knihovny Google Fonts, u každého fontu byly vybrány všechny dostupné znakové sady, každý font byl stažen ve všech variantách tloušťky a kurzívy (Lato, Montserrat, Open Sans, Oswald, Roboto, Robot Condensed, Source Sans Pro)

23 Arabic. *Ghazali* [online]. [cit. 2007-12-10]. Dostupné z: http://www.ghazali.org/arabic/

24 A Review of Chinese Word Lists Accessible on the Internet. *Chih-Hao Tsai's Technology Page* [online]. [cit. 2006-01-01]. Dostupné z: http://technology.chtsai.org/wordlist/

4.3. Výsledky testování

Tabulka 14: Porovnání statistik úspory místa, rychlosti komprese, a rychlosti dekomprese testovaných algoritmů

Algoritmus	Úroveň	Úspora místa (%)			Rychlost komprese (Mbit/s)			Rychlost dekomprese (Mbit/s)		
		min.	průměr	max.	min.	průměr	max.	min.	průměr	max.
DEFLATE	nejnižší	9,11	64,39	92,13	262	636	1 945	487	910	3 323
	nejvyšší	9,53	69,94	96,36	7,97	123	368	569	1 185	6 775
LZMA2	nejnižší	9,42	68,98	95,79	49,4	130	432	121	417	1 699
	nejvyšší	9,84	73,83	97,27	6,62	20	61,3	78,6	414	1 899
LZ4	rychlá	8,36	51,67	88,68	1 559	3 710	14 899	6 940	45 982	85 208
	kvalitní	9,04	62,68	95,79	17,4	260	689	10 975	50 138	90 972
Zopfli	15 iterací	10,12	71,58	96,50	0,12	1,33	2,74	627	1 247	5 011
Brotli	úroveň 0	7,62	63,30	91,29	660	1 577	4 145	423	1 770	4 028
	úroveň 1	8,42	65,00	92,24	581	1 304	3 867	619	1 852	4 672
	úroveň 2	9,65	68,28	92,41	192	653	1 709	664	2 100	5 789
	úroveň 3	9,72	68,77	94,51	348	600	1 776	625	2 251	8 372
	úroveň 4	10,00	69,65	95,62	145	440	1 595	667	2 557	10 713
	úroveň 5	10,14	71,40	96,82	59	233	999	625	2 562	9 748
	úroveň 6	10,14	71,76	97,06	39	193	809	514	2 624	11 058
	úroveň 7	10,13	72,07	97,28	17	121	614	454	2 689	13 317
	úroveň 8	10,13	72,21	97,40	14	90,7	488	635	2 675	12 272
	úroveň 9	10,16	72,32	97,50	12,5	65,1	390	667	2 706	14 013
	úroveň 10	10,78	74,43	97,53	5,44	11	30,9	516	2 228	12 860
	úroveň 11	10,83	74,99	97,72	2,59	4,52	7,41	524	2 399	13 538

V každém sloupci jsou označeny 4 nejlepší výsledky zeleným pozadím, a 4 nejhorší výsledky červeným pozadím. Shrňme si to, co lze ze statistik usoudit:

- Nejvyšší úrovně Brotli (10-11) jsou ideální pro kompresi statických webových zdrojů, mají ze všech testovaných algoritmů nejvyšší úsporu místa
- Průměrné úrovně Brotli (5-9) komprimují lépe a dekomprimují rychleji než většina ostatních algoritmů, a dostatečnou rychlostí komprese i pro dynamické webové zdroje
- Nízké úrovně Brotli (0-4) jsou úsporou místa srovnatelné s DEFLATE, ale mají znatelně vyšší rychlost komprese i dekomprese
- LZMA2 je velice efektivní v úspoře místa, má ovšem nejnižší rychlost dekomprese ze všech testovaných algoritmů (LZMA2 byl kandidát pro kompresi fontů WOFF 2.0, ale velmi nízká

rychlost dekomprese a také vysoké využití paměti byly některými z důvodů proč byl nakonec použit Brotli[25])

- LZ4 jednoznačně vítězí v rychlosti dekomprese, s průměrně 25x vyšší rychlostí dekomprese v porovnání s ostatními algoritmy, nejvyšší úroveň komprese však vykazuje horší výsledky úspory místa a rychlosti komprese než nejnižší úroveň DEFLATE
- Zopfli je nejpomalejší testovaný kompresní algoritmus, jeho hlavní výhoda kompatibility s DEFLATE je zde však znát – díky vyšší úspoře místa jsou komprimované soubory také dekomprimovány rychleji, je tedy stále vhodný pro statické webové zdroje v případech kdy webový prohlížeč nebo server nepodporuje Brotli, nebo formát souboru obsahuje DEFLATE data (PNG, WOFF 1.0)

4.3.1. Brotli ve webovém prostředí

Před rozšířením Brotli byl jedinou volbou komprese webových zdrojů algoritmus DEFLATE, většinou zabalen ve formátu gzip. K dubnu 2018 používá cca 84 % uživatelů webový prohlížeč s podporou dekomprese Brotli,[26] a ve statistikách vidíme nejen 2x rychlejší dekompresi, ale i větší rychlost a efektivitu v kompresi při zvolení vhodné úrovně.

Všechny nejpoužívanější HTTP servery již podporují Brotli, ať už oficiálně nebo ve formě plug-inů. Při analýze podpory Brotli na 5000 webech ze seznamu Alexa Top Sites[27] celkem 1692 webů neodpovědělo, a ze zbývajících 3308 byla podpora zjištěna u 612 webů, tedy přibližně 18,5 %.

Podle zdroje HTTP Archive, který sleduje statistiky o webových stránkách, byla v roce 2017 průměrná velikost HTML, JS, a CSS souborů dohromady 450,5 KiB na jedné stránce, s největšími stránkami dosahujícími až 1400,5 KiB.[28] S průměrnou rychlostí dekomprese Brotli na testovacím počítači 2367,75 Mbit/s jde o zanedbatelných 1,6 ms a 4,9 ms, na mobilních zařízení však může být vyšší rychlost dekomprese Brotli v porovnání s DEFLATE znát.

Fonty ve formátu WOFF

Podívejme se také na statistiky z porovnání 36 fontů ve formátu WOFF 1.0 (DEFLATE) a 2.0 (Brotli):

Tabulka 15: Statistiky srovnávající verze 1.0 a 2.0 formátu fontů WOFF

	Minimální velikost	**Průměrná velikost**	**Maximální velikost**
WOFF 1.0	22,66 KiB	72,30 KiB	111,25 KiB
WOFF 2.0	17,66 KiB	51,74 KiB	84,70 KiB

Průměrná úspora místa je 26,59 %, s minimem 18,22 % a maximem 38,45 %.

25 WOFF 2.0, the inside scoop. *W3C* [online]. [cit. 2018-03-01]. Dostupné z: https://www.w3.org/blog/2018/03/woff-2-0-the-inside-scoop

26 *Can I use...: Browser support tables for modern web technologies* [online]. [cit. 2018-04-20]. Dostupné z: https://caniuse.com/#search=brotli

27 http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

28 Report: Page Weight. *HTTP Archive* [online]. [cit. 2018-01-01]. Dostupné z: https://httparchive.org/reports/page-weight?start=2017_01_01&end=2018_01_01

4.3.2. Úspora místa

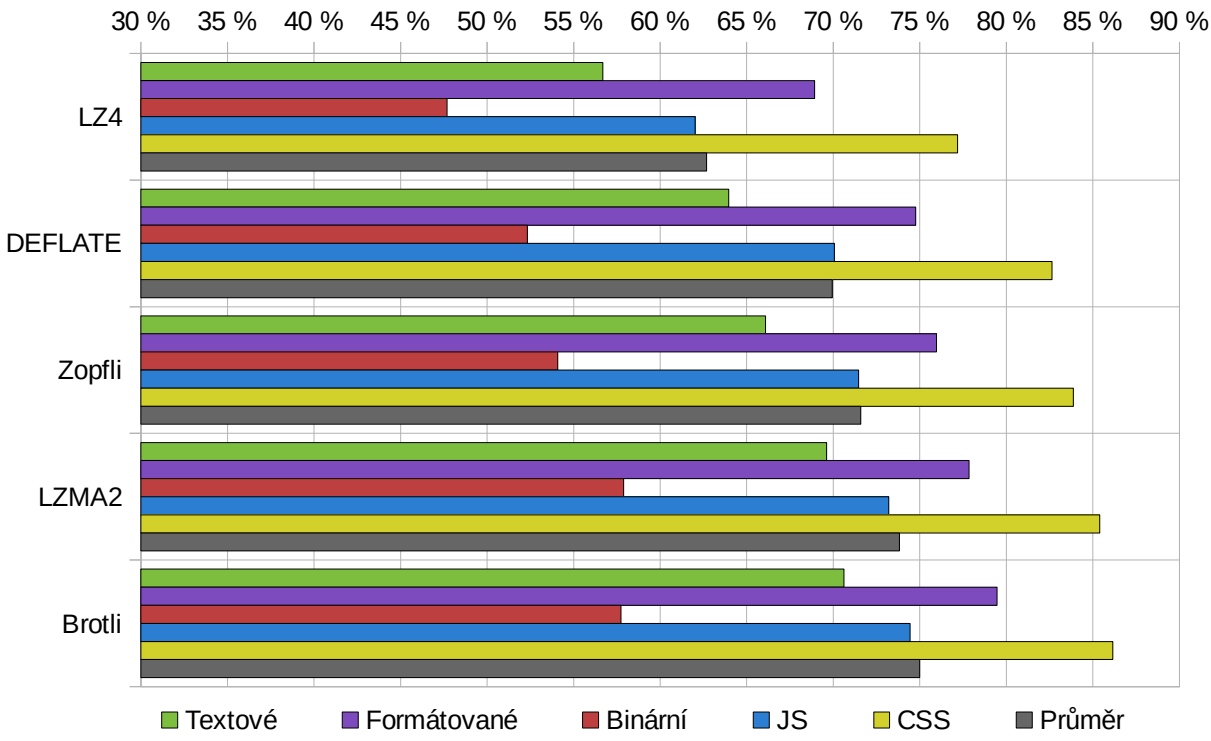
Podíváme-li se na porovnání úspory místa jednotlivých souborů, Brotli vítězí ve většině případů:

- Brotli (úroveň 11) 106 souborů (91,4 %)
- LZMA2 (nejvyšší) 8 souborů (6,9 %)
- Brotli (úroveň 10) 2 soubory (1,7 %)

U dvou souborů kde zvítězila úroveň 10 byl kompresní poměr vyšší o 0,27 %, a u souborů kde zvítězil algoritmus LZMA2 byla kompresní poměr vyšší o 0,92 %.

Podívejme se na průměrnou úsporu místa v závislosti na typech souborů. U každého algoritmu byla použita nejvyšší úroveň komprese (pozn.: algoritmy jsou seřazeny podle průměrné úspory místa).

- Řada *Textové* zahrnuje čistě textové soubory (tzv. *plaintext*) ve všech testovaných mluvených jazycích, tedy nezahrnuje počítačové jazyky
- Řada *Formátované* zahrnuje textové soubory s informacemi o formátování, konkrétně části korpusů Canterbury a Silesia, a soubory ve formátu HTML, XML, a RTF
- Řada *Binární* zahrnuje soubory obsahující především binární data, konkrétně část korpusu Canterbury, a několik PDF souborů které byly z velké části tvořeny komprimovanými daty
- Řady *JS* a *CSS* zahrnují minifikované i neminifikované zdrojové soubory
- Řada *Průměr* zahrnuje všech 116 testovaných souborů



Ilustrace 1: Srovnání průměrné úspory místa nejvyšších úrovní algoritmů podle typu souborů.

4.3.3. Analýza meta-bloků

Tabulka 16: Shrnutí statistik o množství jednotlivých typů meta-bloků v testovaných souborech.

Úroveň Brotli	Komprimované		Nekomprimované		Prázdné	
	průměr	maximum	průměr	maximum	průměr	maximum
úroveň 0	9,27	156	0,28	21	1	1
úroveň 1	9,18	156	0,21	13	1	1
úroveň 2	9,66	155	0,02	2	0	0
úroveň 3	9,38	152	0,02	2	0	0
úroveň 4	1	1	0	0	0	0
úroveň 5	1	1	0	0	0	0
úroveň 6	1	1	0	0	0	0
úroveň 7	1	1	0	0	0	0
úroveň 8	1	1	0	0	0	0
úroveň 9	1	1	0	0	0	0
úroveň 10	1,01	2	0	0	0	0
úroveň 11	1,01	2	0	0	0	0

Připomeňme si, že každý meta-blok je schopen vygenerovat až 16 MiB dekomprimovaných dat, a největší testovaný soubor má velikost 9,72 MiB.

U **komprimovaných meta-bloků** vidíme, že úrovně 0-3 je generují mnohem častěji než ostatní úrovně, zatímco úrovně 4-11 se snaží uložit co nejvíce dat do jednoho meta-bloku. Důvodem je rozdělení kategorií kódů příkazu *insert&copy* (literály, insert&copy délky, vzdálenosti) v každém meta-bloku na skupiny, zatímco pro úrovně 0-3 se kompresor snaží v každém-metabloku limitovat počet příkazů a literálů přibližně na 12 287[29] (konstanta 2FFF v hexadecimální soustavě) což vede k mnohem většímu počtu meta-bloků.

Nekomprimované meta-bloky jsou vytvářeny v případech, kdy rozdíl mezi velikostí nekomprimovaných a komprimovaných dat je příliš malý, což je u nižších úrovní komprese pravděpodobnější. V tomto případě jsou však statistiky zkreslené, jelikož byly všechny vygenerovány ve velmi malém počtu souborů, a to konkrétně:

- Úrovně 0-1 vygenerovaly nekomprimované meta-bloky ve 3 PDF souborech
- Úrovně 2-3 vygenerovaly nekomprimované meta-bloky v 1 PDF souboru

Prázdné meta-bloky byly u testovaných dat použity pouze úrovněmi 0 a 1. Tyto úrovně používají velmi jednoduchý algoritmus komprese který na konci souboru vždy vygeneruje meta-blok který je označen jako poslední a prázdný, což mírně zjednodušuje kompresi v případech kdy jsou vygenerovány nekomprimované meta-bloky které nemohou být označeny jako poslední.

29 Brotli. *GitHub* [online]. [cit. 2018-04-13]. Dostupné z: https://github.com/google/brotli/blob/68db5c/c/enc/encode.c#L1068-L1070

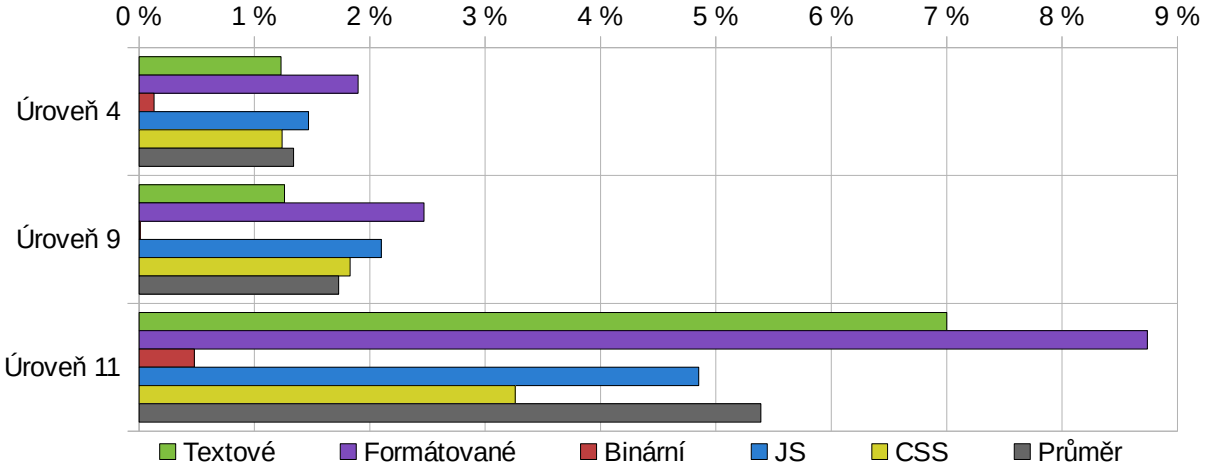
4.3.4. Analýza využití statického slovníku

Tabulka 17: Shrnutí statistik o využití a efektivity statického slovníku Brotli v testovaných souborech.

Úroveň Brotli	Počet slov		Délka slova	Celkem vygenerováno bajtů		Poměr k velikosti souboru	
	průměr	maximum	průměr	průměr	maximum	průměr	maximum
úroveň 0	0	0	0	0	0	0 %	0 %
úroveň 1	0	0	0	0	0	0 %	0 %
úroveň 2	7,9	353	5,5	57,7	2 608	0,01 %	0,23 %
úroveň 3	0	0	0	0	0	0 %	0 %
úroveň 4	256,3	3 567	9,4	1 848	28 421	1,34 %	11,47 %
úroveň 5	318,6	2 592	6,7	1 979	18 233	1,77 %	14,27 %
úroveň 6	299,8	1 652	6,7	1 848	11 337	1,75 %	14,27 %
úroveň 7	285,8	1 032	6,6	1 751	6 767	1,73 %	14,36 %
úroveň 8	283,6	954	6,6	1 736	6 388	1,73 %	14,36 %
úroveň 9	282,4	949	6,6	1 729	6 380	1,73 %	14,27 %
úroveň 10	986,4	12 686	7,3	9 923	149 990	4,97 %	29,41 %
úroveň 11	1 131,3	11 808	7,0	10 322	130 552	5,39 %	29,60 %

Sloupec *délka slova* zahrnuje pouze samotné slovo bez prefixu nebo sufixu, sloupce *celkem vygenerováno bajtů* a *poměr k velikosti souboru* zahrnují i prefixy a sufixy.

Pro zajímavost se můžeme podívat na průměrný poměr k velikosti souboru v závislosti na jeho typu:



Ilustrace 2: Srovnání průměrného poměru dat generovaných statickým slovníkem k originální velikosti souborů podle typu souborů.

4.3.5. Analýza kódů příkazu insert&copy

Tabulka 18: Shrnutí statistik o množství skupin kategorií kódů příkazu insert&copy na meta-blok pro jednotlivé úrovně komprese.

Úroveň Brotli	Literály		Insert&copy délky		Vzdálenosti	
	průměr	maximum	průměr	maximum	průměr	maximum
úroveň 0-3	1	1	1	1	1	1
úroveň 4	5,3	167	1,7	9	8,9	37
úroveň 5	3,4	152	1,7	10	8,4	34
úroveň 6	3,4	153	1,9	9	9,0	38
úroveň 7	3,4	154	2,0	9	9,9	45
úroveň 8	3,4	154	2,0	9	10,3	51
úroveň 9	3,3	155	2,0	10	10,6	54
úroveň 10	7,3	256	3,2	19	5,7	29
úroveň 11	5,6	256	3,7	17	6,4	37

Tabulka 19: Shrnutí statistik o množství Huffmanových kódů kategorií příkazu insert&copy na meta-blok pro jednotlivé úrovně komprese.

Úroveň Brotli	Literály		Insert&copy délky		Vzdálenosti	
	průměr	maximum	průměr	maximum	průměr	maximum
úroveň 0-3	1	1	1	1	1	1
úroveň 4	5,3	167	1,7	9	8,9	37
úroveň 5	3,5	152	1,7	10	8,4	34
úroveň 6	3,6	153	1,9	9	9,0	38
úroveň 7	4,2	154	2,0	9	9,9	45
úroveň 8	4,2	154	2,0	9	10,3	51
úroveň 9	4,0	155	2,0	10	10,6	54
úroveň 10	20,4	256	3,2	19	7,3	33
úroveň 11	15,2	256	3,7	17	7,4	38

Úrovně 10 a 11 používají jiný algoritmus pro rozdělování skupin, je zde například vidět velký rozdíl v počtu skupin vzdáleností nebo v Huffmanových kódech pro literály, v porovnání s úrovněmi 4-9.

Připomeňme si také, že insert&copy délky nepoužívají kontextové modelování, literály jej používají až od úrovně 5, a vzdálenosti jej používají jen v úrovních 10 a 11. Žlutým pozadím je v tabulkách vyznačena přítomnost kontextového modelování, pro všechny neoznačené úrovně a kategorie je pak samozřejmě počet skupin shodný s počtem Huffmanových kódů.

Kontextové módy pro literály

Od úrovně 5 má každá skupina literálů má definován vlastní kontextový mód, aktuální verze kompresoru však vždy použije stejný kontextový mód pro všechny skupiny v meta-bloku.[30]

Úrovně 5-9 pro všechny testované soubory použily mód UTF8, úrovně 10-11 použily mód UTF8 pro téměř všechny soubory – pro 6 souborů byl použit mód Signed:

- 3 soubory z korpusu Canterbury (kennedy.xls, ptt5, sum) které obsahují z velké části nulové bajty a obecně bajty mimo rozsah viditelných znaků kódování UTF-8
- 3 PDF soubory z Projektu Gutenberg které obsahují komprimovaná data namísto čistého PostScriptu v textové formě

Současná verze kompresoru vybírá kontextový mód pomocí jednoduché heuristiky, kdy mód UTF8 je použit v naprosté většině případů, a mód Signed je použit jen úrovních 10-11 pro soubory jejichž obsah kompresor neidentifikuje jako znaky kódování UTF-8. Ostatní módy zatím sám použít neumí, jsou nebo byly však podle vývojářů využity v kompresoru WOFF 2.0.[31]

Jedná se o potenciální oblast budoucího vývoje, jelikož je zde možnost zlepšení komprese binárních dat, textových dat s velkým množstvím speciálních znaků kde může být mód LSB6 uplatněn, a také kombinovaných dat kde může být pro každou skupinu literálů použit jiný kontextový mód.

30 Brotli. *GitHub* [online]. [cit. 2018-04-13]. Dostupné z: https://github.com/google/brotli/blob/68db5c/c/enc/metablock.c#L203-L205

31 ALAKUIJALA, Jyrki. New Compression Codecs Risk Making Zlib Obsolete. In: *Encode's Forum* [online]. [cit. 2016-01-19]. Dostupné z: https://encode.ru/threads/2420-New-Compression-Codecs-Risk-Making-Zlib-Obsolete?p=46259&viewfull=1#post46259

5. Závěr

Prvním cílem této práce byl popis Brotli a algoritmu dekomprese, s implementací dekomprese v konzolové aplikaci psané v jazyce C#. Pomocí oficiální specifikace formátu Brotli, dokumentu RFC 7932, byla vyvinuta aplikace s možností dekomprese souboru a výpisu informací o jeho formátu.

Druhým cílem bylo srovnání Brotli s dalšími kompresními metodami, a ověření jeho vhodnosti pro použití na webu a nahrazení stávající metody DEFLATE. Ke srovnání byl vybrán soubor primárně textových testovacích dat, tvořen ze známých korpusů nad kterými se komprese typicky testuje, literárních děl a dokumentů v několika světových jazycích, a webových zdrojů ve formátu HTML, CSS, a JS. Konzolová aplikace pak s pomocí nativních knihoven testovaných algoritmů pro kompresi a dekompresi provedla nad těmito daty testy efektivity a výkonu. Z testů vyplynulo, že Brotli je, se správným nastavením, nejvhodnější kompresní metodou pro textové statické i dynamické webové zdroje.

Třetím cílem bylo vysvětlení a analýza jednotlivých kompresních úrovní Brotli. K tomu byla použita vlastní implementace dekomprese v aplikaci, do které byla přidána funkce sběru a výpisu informací o formátu souboru, se kterou byly nad stejným souborem dat provedeny testy.

Při tvorbě práce bylo nalezeno několik podnětů k dalšímu potenciálnímu výzkumu a vývoji Brotli. Jedním z nich je rozšíření funkcionality a efektivity kompresoru, například v oblasti kontextového modelování jehož potenciál aktuálně není plně využit, a další vývoj v této oblasti má možnost zlepšení efektivity komprese netextových dat, a kombinovaných textových a binárních dat. Dále je kupříkladu možný výzkum optimalizace současného algoritmu komprese, aby mohly být vyšší úrovně komprese použity i pro dynamický webový obsah s rozumnou rychlostí.

Literatura

1. ALAKUIJALA, Jyrki a Zoltan SZABADKA. RFC 7932. *IETF Tools* [online]. 2016 [cit. 2018-04-24]. Dostupné z: https://tools.ietf.org/html/rfc7932
2. CHOUDHARY, Suman M., Anjali S. PATEL a Sonal J. PARMAR. Study of LZ77 and LZ78 Data Compression Techniques. *International Journal of Engineering Science and Innovative Technology* [online]. 2015, **4**(3) [cit. 2018-04-24]. ISSN 2319-5967. Dostupné z: https://pdfs.semanticscholar.org/e087/085514d51ed377eec77b7a75c211d8739231.pdf
3. MCCLOSKEY, Robert. Canonical Huffman Coding. *CMPS 340* [online]. 2015 [cit. 2018-04-24]. Dostupné z: http://www.cs.uofs.edu/~mccloske/courses/cmps340/huff_canonical_dec2015.html
4. ALAKUIJALA, Jyrki, Evgenii KIUCHNIKOV, Zoltan SZABADKA a Lode VANDEVENNE. Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms. *GitHub* [online]. Google, 2015 [cit. 2015-09-22]. Dostupné z: https://github.com/google/brotli/blob/master/docs/brotli-comparison-study-2015-09-22.pdf
5. DEUTSCH, Peter. RFC 1951. *IETF Tools* [online]. 1996 [cit. 2018-04-12]. Dostupné z: https://tools.ietf.org/html/rfc1951
6. 7z Format. *7-Zip* [online]. c2018 [cit. 2018-04-12]. Dostupné z: https://www.7-zip.org/7z.html
7. The .xz File Format. *The Tukaani Project* [online]. [cit. 2009-08-27]. Dostupné z: https://tukaani.org/xz/xz-file-format.txt
8. *LZ4* [online]. [cit. 2018-04-12]. Dostupné z: http://lz4.github.io/lz4
9. Zopfli. *GitHub* [online]. [cit. 2017-07-07]. Dostupné z: https://github.com/google/zopfli
10. Brotli. *GitHub* [online]. [cit. 2018-04-13]. Dostupné z: https://github.com/google/brotli
11. *The Canterbury Corpus* [online]. [cit. 2001-01-08]. Dostupné z: http://corpus.canterbury.ac.nz/purpose.html
12. *Project Gutenberg* [online]. [cit. 2018-04-10]. Dostupné z: https://www.gutenberg.org
13. Arabic. *Ghazali* [online]. [cit. 2007-12-10]. Dostupné z: http://www.ghazali.org/arabic/
14. A Review of Chinese Word Lists Accessible on the Internet. *Chih-Hao Tsai's Technology Page* [online]. [cit. 2006-01-01]. Dostupné z: http://technology.chtsai.org/wordlist/
15. WOFF 2.0, the inside scoop. *W3C* [online]. [cit. 2018-03-01]. Dostupné z: https://www.w3.org/blog/2018/03/woff-2-0-the-inside-scoop
16. *Can I use...: Browser support tables for modern web technologies* [online]. [cit. 2018-04-20]. Dostupné z: https://caniuse.com/#search=brotli
17. Report: Page Weight. *HTTP Archive* [online]. [cit. 2018-01-01]. Dostupné z: https://httparchive.org/reports/page-weight?start=2017_01_01&end=2018_01_01
18. ALAKUIJALA, Jyrki. New Compression Codecs Risk Making Zlib Obsolete. In: *Encode's Forum* [online]. [cit. 2016-01-19]. Dostupné z: https://encode.ru/threads/2420-New-Compression-Codecs-Risk-Making-Zlib-Obsolete?p=46259&viewfull=1#post46259

Seznam příloh

Příloha I. Transformace statického slovníku
Seznam všech 121 transformací statického slovníku s prefixy, sufixy, a použitými transformačními funkcemi. Příloha je tištěna na 2 stranách.

Příloha II. Tabulky kontextových módů
Tabulky použité pro definici funkcí kontextových módů pro literály UTF8 a Signed. Příloha je tištěna na 2 stranách.

Příloha III. Zdrojový kód aplikace
Příloha je ve složce *Zdrojovy kod* na CD, obsahuje zdrojový kód a soubory projektu pro Visual Studio 2017.

Příloha IV. Sestavená aplikace
Příloha je ve složce *Aplikace* na CD, obsahuje 64bitové sestavení konzolové aplikace a knihoven.

Příloha V. Výsledky testů a skripty
Příloha je ve složce *Testovani* na CD, obsahuje PowerShell a Python skripty použité pro testování a generování výsledků, a také všechny tabulky s daty.

Příloha I. Transformace statického slovníku

Text prefixů a sufixů je v uvozovkách, aby byly zřejmé pozice mezer. Zpětné lomítko je zde použito pro vyjádřeních speciálních a neviditelných znaků ve stylu tzv. escape sekvencí jazyka C.

RFC 7932, Appendix B. List of Word Transformations: https://tools.ietf.org/html/rfc7932#appendix-B

Index	Prefix slova	Transform. funkce	Sufix slova
0	""	Identity	""
1	""	Identity	" "
2	" "	Identity	" "
3	""	OmitFirst1	""
4	""	FermentFirst	" "
5	""	Identity	" the "
6	" "	Identity	""
7	"s "	Identity	" "
8	""	Identity	" of "
9	""	FermentFirst	""
10	""	Identity	" and "
11	""	OmitFirst2	""
12	""	OmitLast1	""
13	", "	Identity	" "
14	""	Identity	", "
15	" "	FermentFirst	" "
16	""	Identity	" in "
17	""	Identity	" to "
18	"e "	Identity	" "
19	""	Identity	"\""
20	""	Identity	"."
21	""	Identity	"\">"
22	""	Identity	"\n"
23	""	OmitLast3	""
24	""	Identity	"]"
25	""	Identity	" for "
26	""	OmitFirst3	""
27	""	OmitLast2	""

Index	Prefix slova	Transform. funkce	Suffix slova
28	""	Identity	" a "
29	""	Identity	" that "
30	" "	FermentFirst	""
31	""	Identity	". "
32	"."	Identity	""
33	" "	Identity	", "
34	""	OmitFirst4	""
35	""	Identity	" with "
36	""	Identity	"'"
37	""	Identity	" from "
38	""	Identity	" by "
39	""	OmitFirst5	""
40	""	OmitFirst6	""
41	" the "	Identity	""
42	""	OmitLast4	""
43	""	Identity	". The "
44	""	FermentAll	""
45	""	Identity	" on "
46	""	Identity	" as "
47	""	Identity	" is "
48	""	OmitLast7	""
49	""	OmitLast1	"ing "
50	""	Identity	"\n\t"
51	""	Identity	":"
52	" "	Identity	". "
53	""	Identity	"ed "
54	""	OmitFirst9	""
55	""	OmitFirst7	""

56	""	OmitLast6	""
57	""	Identity	"("
58	""	FermentFirst	", "
59	""	OmitLast8	""
60	""	Identity	" at "
61	""	Identity	"ly "
62	" the "	Identity	" of "
63	""	OmitLast5	""
64	""	OmitLast9	""
65	" "	FermentFirst	", "
66	""	FermentFirst	"\""
67	"."	Identity	"("
68	""	FermentAll	" "
69	""	FermentFirst	"\">"
70	""	Identity	"=\""
71	" "	Identity	"."
72	".com/"	Identity	""
73	" the "	Identity	" of the "
74	""	FermentFirst	"""
75	""	Identity	". This "
76	""	Identity	","
77	"."	Identity	" "
78	""	FermentFirst	"("
79	""	FermentFirst	"."
80	""	Identity	" not "
81	" "	Identity	"=\""
82	""	Identity	"er "
83	" "	FermentAll	" "
84	""	Identity	"al "
85	" "	FermentAll	""
86	""	Identity	"='"
87	""	FermentAll	"\""
88	""	FermentFirst	". "

89	" "	Identity	"("
90	""	Identity	"ful "
91	" "	FermentFirst	". "
92	""	Identity	"ive "
93	""	Identity	"less "
94	""	FermentAll	"""
95	""	Identity	"est "
96	" "	FermentFirst	"."
97	""	FermentAll	"\">"
98	" "	Identity	"='"
99	""	FermentFirst	","
100	""	Identity	"ize "
101	""	FermentAll	"."
102	"\xc2\xa0"	Identity	"""
103	" "	Identity	","
104	""	FermentFirst	"=\""
105	""	FermentAll	"=\""
106	""	Identity	"ous "
107	""	FermentAll	", "
108	""	FermentFirst	"='"
109	" "	FermentFirst	","
110	" "	FermentAll	"=\""
111	" "	FermentAll	", "
112	""	FermentAll	","
113	""	FermentAll	"("
114	""	FermentAll	". "
115	" "	FermentAll	"."
116	""	FermentAll	"='"
117	" "	FermentAll	". "
118	" "	FermentFirst	"=\""
119	" "	FermentAll	"='"
120	" "	FermentFirst	"='"

Příloha II. Tabulky kontextových módů

Označme poslední zapsaný bajt *p1*, předposlední zapsaný bajt *p2*, a definujme operace:

- `x | y` bitová operace OR
- `x << y` posunutí bitů hodnoty *x* doleva o *y* pozic
- `t[n]` přečtení *n*-tého prvku tabulky *t*

Funkce módu *UTF8* je definována jako: `LUT_0[p1] | LUT_1[p2]`
Funkce módu *Signed* je definována jako: `(LUT_2[p1] << 3) | LUT_2[p2]`

RFC 7932, Context Modes and Context ID Lookup for Literals:
https://tools.ietf.org/html/rfc7932#section-7.1

LUT_0

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0x	0	0	0	0	0	0	0	0	0	4
1x	4	0	0	4	0	0	0	0	0	0
2x	0	0	0	0	0	0	0	0	0	0
3x	0	0	8	12	16	12	12	20	12	16
4x	24	28	12	12	32	12	36	12	44	44
5x	44	44	44	44	44	44	44	44	32	32
6x	24	40	28	12	12	48	52	52	52	48
7x	52	52	52	48	52	52	52	52	52	48
8x	52	52	52	52	52	48	52	52	52	52
9x	52	24	12	28	12	12	12	56	60	60
10x	60	56	60	60	60	56	60	60	60	60
11x	60	56	60	60	60	60	60	56	60	60
12x	60	60	60	24	12	28	12	0	0	1
13x	0	1	0	1	0	1	0	1	0	1
14x	0	1	0	1	0	1	0	1	0	1
15x	0	1	0	1	0	1	0	1	0	1
16x	0	1	0	1	0	1	0	1	0	1
17x	0	1	0	1	0	1	0	1	0	1
18x	0	1	0	1	0	1	0	1	0	1
19x	0	1	2	3	2	3	2	3	2	3
20x	2	3	2	3	2	3	2	3	2	3
21x	2	3	2	3	2	3	2	3	2	3
22x	2	3	2	3	2	3	2	3	2	3
23x	2	3	2	3	2	3	2	3	2	3
24x	2	3	2	3	2	3	2	3	2	3
25x	2	3	2	3	2	3				

LUT_1

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0x	0	0	0	0	0	0	0	0	0	0
1x	0	0	0	0	0	0	0	0	0	0
2x	0	0	0	0	0	0	0	0	0	0
3x	0	0	0	1	1	1	1	1	1	1
4x	1	1	1	1	1	1	1	1	2	2
5x	2	2	2	2	2	2	2	2	1	1
6x	1	1	1	1	1	2	2	2	2	2
7x	2	2	2	2	2	2	2	2	2	2
8x	2	2	2	2	2	2	2	2	2	2
9x	2	1	1	1	1	1	1	3	3	3
10x	3	3	3	3	3	3	3	3	3	3
11x	3	3	3	3	3	3	3	3	3	3
12x	3	3	3	1	1	1	1	0	0	0
13x	0	0	0	0	0	0	0	0	0	0
14x	0	0	0	0	0	0	0	0	0	0
15x	0	0	0	0	0	0	0	0	0	0
16x	0	0	0	0	0	0	0	0	0	0
17x	0	0	0	0	0	0	0	0	0	0
18x	0	0	0	0	0	0	0	0	0	0
19x	0	0	0	0	0	0	0	0	0	0
20x	0	0	0	0	0	0	0	0	0	0
21x	0	0	0	0	0	0	0	0	0	0
22x	0	0	0	0	2	2	2	2	2	2
23x	2	2	2	2	2	2	2	2	2	2
24x	2	2	2	2	2	2	2	2	2	2
25x	2	2	2	2	2	2				

LUT_2

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0x	0	1	1	1	1	1	1	1	1	1
1x	1	1	1	1	1	1	2	2	2	2
2x	2	2	2	2	2	2	2	2	2	2
3x	2	2	2	2	2	2	2	2	2	2
4x	2	2	2	2	2	2	2	2	2	2
5x	2	2	2	2	2	2	2	2	2	2
6x	2	2	2	2	3	3	3	3	3	3
7x	3	3	3	3	3	3	3	3	3	3
8x	3	3	3	3	3	3	3	3	3	3
9x	3	3	3	3	3	3	3	3	3	3
10x	3	3	3	3	3	3	3	3	3	3
11x	3	3	3	3	3	3	3	3	3	3
12x	3	3	3	3	3	3	3	3	4	4
13x	4	4	4	4	4	4	4	4	4	4
14x	4	4	4	4	4	4	4	4	4	4
15x	4	4	4	4	4	4	4	4	4	4
16x	4	4	4	4	4	4	4	4	4	4
17x	4	4	4	4	4	4	4	4	4	4
18x	4	4	4	4	4	4	4	4	4	4
19x	4	4	5	5	5	5	5	5	5	5
20x	5	5	5	5	5	5	5	5	5	5
21x	5	5	5	5	5	5	5	5	5	5
22x	5	5	5	5	5	5	5	5	5	5
23x	5	5	5	5	5	5	5	5	5	5
24x	6	6	6	6	6	6	6	6	6	6
25x	6	6	6	6	6	7				